



# No-Code Programming for Biology

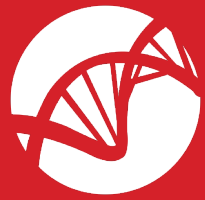
# Rapid Prototyping

Beginner's Guide  
Second Edition

Biomaker Team  
University of Cambridge







# Biomaker

# No-Code Programming for Biology

# Rapid Prototyping

Beginner's Guide 2<sup>nd</sup> edition

Jim Haseloff, Stephanie Norwood & Matt Wayland

Learn how to:

- Understand and control an Arduino board
- Program without using code
- Use simple electronic components such as screens and sensors
- Build your own devices for use in biological research

ISBN-978-1-7394029-0-7

Published 2023





## Biomaker Initiative

Biomaker was started in 2014 as an interdisciplinary scheme for project-based learning and innovation, was founded by Jim Haseloff as part of the Engineering Biology Interdisciplinary Research Centre at the University of Cambridge and OpenPlant, one of the UK's six National Synthetic Biology Research Centres. It has been funded mainly by contributions from the UKRI research councils in the UK, from BBSRC, EPSRC and NERC grant programmes. It has also received sponsorship for workshop materials from Sseed Studio.

Biomaker has provided funding for interdisciplinary team-based projects at the intersection of electronics, computer science, 3D printing, sensor technology, low cost DIY instrumentation and biology, as well as workshops and outreach events. The initiative aims to build open technologies and promote development of research skills and collaborations. It taps into existing open standards and a rich ecosystem of resources for microcontrollers, first established to simplify programming and physical computing for designers, artists and scientists. These tools allow biologists to program and develop real-world laboratory tools. The Biomaker project also provides a direct route for physical scientists and engineers to get hands-on experience with biological systems.

We aim to lower the barriers that impede interdisciplinary work, and to promote the kinds of training that are useful for building instruments and devices for biological experiments in the lab and field. We develop starter kits for no-code programming that allow biologists to build bioinstrument prototypes for measurement and control of biological systems. These have a wide range of applications including instrumentation, microscopy, microfluidics, 3D printing, biomedical devices, DNA design, plant sciences and outreach and public engagement. You can find examples of documented projects on the Biomaker website at [www.biomaker.org](http://www.biomaker.org).

An important aspect of Biomaker is the use of open source and low cost tools and hardware, which facilitate equitable access to fundamental knowledge and technology, encourage a collaborative environment, and support the establishment of an open, sustainable bioeconomy.





# Welcome to No-Code Programming for Biology

This second edition of the Beginner's Guide to Rapid Prototyping has been put together by the Biomaker team to help you get to grips with the basics of rapid prototyping and building custom instrumentation for biological research.

We have extended the no-code programming tutorial material laid out in the first edition by Steph Norwood, and included more material on hardware expansion and building prototypes.

Designed for readers who might have little or no experience with text-based coding or hardware, this guide makes use of free open-source software and low-cost hardware to introduce you the principles behind making your own instruments.

Working though this guide can be useful as a base for those with a specific challenge or task in mind, as well as for those who are simply looking to expand their skillsets for experimental design.

While we focus on learning aspects that are useful for biological research, the

information in this guide can also be used for a wide range of no-code programming applications, and we hope that these skills can be applicable whatever your area of interest.

The guide will teach you how to use the free open-source, no-code programming software XOD, as well as how to use some simple low-cost hardware devices, such as LEDs, sensors and screens.

The training material is built to accompany the integrated Grove All-In-One Beginner Kit for Arduino development board, designed by Seeed Studio. At the time of writing, the kit is available from global stockists for approximately £20/\$25. Alternative versions of the Arduino Uno board and accompanying components can also be used, but you will need to wire-up these components as you go.

In this 2<sup>nd</sup> edition of the guide, we also describe how to extend this basic set of integrated components with new external devices.

Jim Haseloff.

## AUTHORS

Jim Haseloff  
Stephanie Norwood  
Matt Wayland

## IMAGES

Jim Haseloff  
Stephanie Norwood  
SparkFun Electronics  
Adafruit Industries  
Seed Studio  
Biomaker Challenge Participants

## XOD LIBRARY CREATORS

Matt Wayland  
Marco Aita  
Cesar Sosa  
XOD user: gst  
XOD user: e  
XOD user: gweimer

## CONTACT

Jim Haseloff  
jh295@cam.ac.uk



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

# Contents

<b>Lesson 1: Introduction</b>	
Introduction	2
The Guide	4
The Starter Kit	6
The Microcontroller	8
The XOD IDE	12
<b>Lesson 2: Getting Started</b>	
Getting started	18
Setting up your board	19
Task 1: Testing your board	20
Task 2: Input and Output devices	26
<b>Lesson 3: Explore XOD</b>	
Explore XOD	32
Task 3: Tweak and Watch nodes	33
Task 4: Flip, Clock and Count nodes	36
Task 5: Concat, Join and Format-Number Nodes	42
<b>Lesson 4: Building Devices</b>	
Building devices	48
Task 6: Creating new nodes	49
Task 7: Using Buses	56
Task 8: Program logic	60
Task 9: Sequences and Loops	64
<b>Embedded Hardware</b>	
Embedded hardware	72
XOD nodes for the embedded hardware	74
<b>Lesson 5: Next Steps</b>	
Next Steps	84
Hardware expansion	85
Documenting circuit construction	90
Hardware stands	92
Connecting external hardware	94
Electrical connections	96

<b>External Components</b>	
16x2 character LCD screen	98
Ring of RGB LEDs (WS2812)	102
Real time clock (DS1302)	106
Humidity and light sensor (SHT20)	110
Light sensor (BH1750)	114
Colour sensor (TCS3472)	118
Laser range-finder (VL6180X)	122
Temperature sensor (MCP9808)	126
IR Motion sensor (HC-SR501)	130
Microwave radar proximity sensor (RCWL-0516)	134
Weight sensor (HX711)	138
Water quality sensor (TDS meter 1.0)	142
Water level sensor	146
Soil moisture sensor	150
<b>Software Expansion</b>	
Software expansion	155
Creating a XOD library for the TSL2591 light sensor	156
Testing the Arduino library	158
Creating a device node	160
Device C++ code	162
Documenting the device node	163
Creating a set-timing node	164
Set-timing C++ code	166
Creating a quick-start node	167
Example patches	168
Publishing your library	169
Useful software resources	171
<b>Case Studies</b>	
Case studies	172
Example: the AirFlow reactor	176
Additional Information	190
<b>Glossary</b>	192
<b>Index</b>	196
<b>Acknowledgements</b>	198





# Lesson 1: Introduction



**The Guide**



**The Starter Kit**



**The Microcontroller**



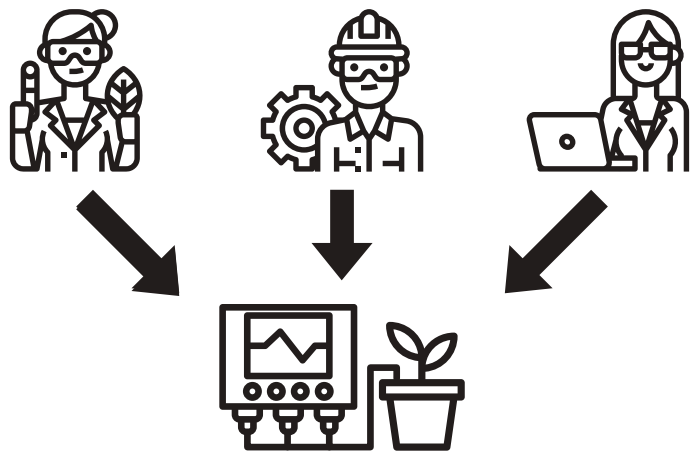
**The XOD IDE**

# Introduction

## Biomaker and No-Code Programming for Biology

The Biomaker team has put together this guide to introduce biologists, or other scientists with little formal programming training, to the basics of Biomaking, including:

- Use of Arduino-based microcontrollers
- Use of sensors, displays and actuators
- Use of XOD visual programming



These new skills can be enabling in many ways. Scientists can gain expertise and new ways of thinking to apply to their work. Moreover, the components for this type of instrumentation are often very cheap, especially when compared with off-the-shelf commercial solutions. The use of simple hardware and software resources allow easy modification, extension and repair of custom instruments, and the use of open-source components and systems promotes sharing of information and set up of collaborative projects. This creates a growing set of resources for the community to draw from, and build upon.

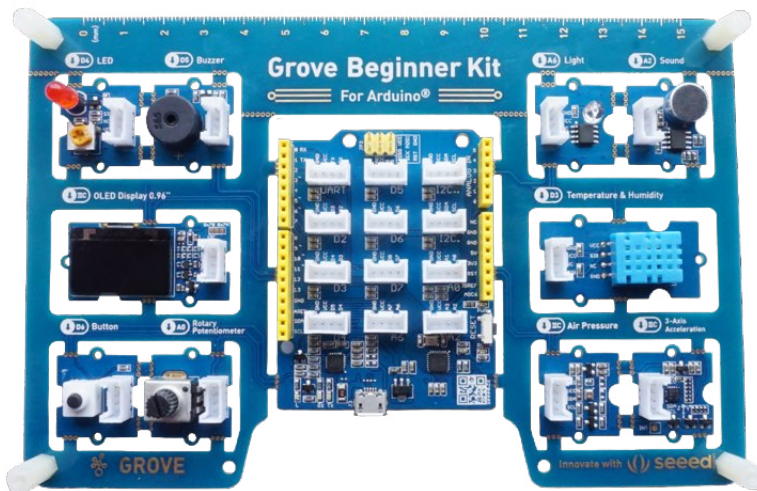
We hope that you enjoy taking part in this online course, that you learn something new and that you find it useful for your future career. Most of all, we encourage anyone who is interested in developing their skills further to sign up for a Biomaker Challenge, where you can join a team of like-minded scientists and engineers to build bioinstruments for real-world applications.



In this first lesson we will cover some of the basic background information you will need to know before you start programming.

First, we will take a look at this guide and how to use it. Then we will look at the Grove board and explore each of its built-in devices, including how they might be used. Next, we will briefly discuss microcontrollers and how to programme them, and finally we will introduce you to the XOD IDE software and some of its terminology.

These basics will help you to become familiar with the tools we will be using throughout this guide. Many of these concepts will be covered again as we apply this knowledge to perform hands-on tasks later in this guide.



*The Grove All-in-One Beginner Kit for Arduino*

## OBJECTIVES

By the end of this chapter you should be able to:

- Name the different parts of the Grove board and give examples of how they might be used.
- Describe the basic concepts of a microcontroller.
- Describe the steps involved in programming the Arduino board and how information is transmitted in this system.
- Name three of the most common types of electronic communication and explain the difference between them.
- List the pin (port) connections for each of the board's components.
- Recall the different parts of the XOD IDE software and describe what each part is used for.
- Recount the three key terms used in XOD programming and what they mean.
- List the data types used in XOD and give examples of each.

# The Guide

The guide is split into four core lessons, each described below. These lessons are designed to be worked through in order, and start by exploring the Biomaker starter kit and the XOD integrated development environment (IDE). The rest of the guide will then take you through a series of tasks designed to introduce you to your board, as well as some key aspects of programming in XOD.

The final chapter provides some additional useful information, as well as some details about how to expand your skills and get started with designing your own devices.

In addition to the information in this guide, the XOD website also provides some useful tutorials and a community forum where you can find help at [www.xod.io](http://www.xod.io)



## Introduction

This chapter will give you a brief introduction to the Biomaker starter kit, including the Grove board, how to control it, and how to use XOD.



## Getting Started

This chapter will take you through a few simple tasks to get started with using your board and the XOD IDE. You'll learn to use the LED, buzzer and button devices.



## Explore XOD

This chapter will explore some of the most useful functions of XOD. Understanding how to use these functions will give you a great base to work from.



## Building Devices

This chapter will delve into some more complex functions in XOD. By the end, you should be able to start developing your own ideas, programmes and devices!

## The No-Code Programming for Biology Handbook

In addition to this beginner's guide, the Biomaker team has also created a range of useful beginner and advanced resources available on the Biomaker website. These resources are designed to help you learn more about the possibilities of Biomaking and to expand your capability to start building your own devices. They include additional tutorials, videos and information about commonly used hardware and expansion devices.

All of the Biomaker and No-Code Programming for Biology resources are available to download on the Biomaker website at [www.biomaker.org/resources](http://www.biomaker.org/resources).





## Tools to Accompany the Guide

### GROVE BEGINNER KIT FOR ARDUINO

The Biomaker starter kit is composed of this beginner's guide, and the Grove Beginner Kit for Arduino. This kit is made by the open source hardware company Seeed Studio and is based on a simple Arduino microcontroller. The kit comes as an integrated PCB board with several useful input and output devices already connected and ready to go. No soldering, wiring or connecting of components means it's perfect for getting started with hardware!

The Starter Kit section provides a quick summary of each part of the board and what it might be used for, whilst the Microcontroller section gives a little background on the Arduino board.

### XOD IDE

The XOD integrated development environment (IDE) is a free open-source software that allows you to programme microcontroller-based devices, such as Grove or Arduino boards, using visual 'nodes' rather than written code. Nodes can represent devices or functions, and by linking them together in different ways you can create a wide variety of different programmes. Programming visually like this can save some of the time and energy required to learn a new language and large amounts of syntax.

The XOD IDE section provides a quick summary of the different parts of the XOD IDE, and what you'll see when you first load the software, as well as some useful terminology used in XOD programming.

### XOD WEBSITE

The XOD website ([www.xod.io](http://www.xod.io)) provides plenty of useful information for beginners, including tutorials and a user guide under the 'Documentation' tab, a database of libraries under the 'Libraries' tab, and a very helpful forum under the 'Community' tab.

### BIOMAKER WEBSITE

The Biomaker website ([www.biomaker.org](http://www.biomaker.org)) hosts a variety of useful materials, including digital downloads of this guide, the accompanying tutorial file, and a number of other Biomaker tutorials and handbooks. These can be found under the 'Getting Started' tab.

You can also find examples of previous Biomaker projects on the website under the 'Projects' tab. With over 180 projects so far, there is plenty of inspiration for the budding Biomaker. Projects are also documented on the Biomaker Hackster Hub ([www.hackster.io/biomaker](http://www.hackster.io/biomaker)).

# The Starter Kit

The Biomaker starter kit is composed of this beginner's guide, and the Grove Beginner Kit for Arduino. This kit is made by the open source hardware company Seeed Studio and is based on a simple Arduino microcontroller.

1

## LED

A red light emitting diode (LED). This light can be used as a notification or warning signal in devices.

2

## BUZZER

Inbuilt piezoelectric buzzer. Can be programmed to emit tones at different frequencies.

3

## OLED SCREEN

High quality OLED screen, which can be used to display both images and text. It is a 64x128 pixel matrix which can display desired content in monochrome (black and white).

4

## BUTTON

A simple button that responds to user input (presses). Can be used as an on/off switch or trigger.

5

## ROTARY POTENTIOMETER

Also known as a knob sensor as it senses the rotation angle of the knob. Can be used as a dial to change volume or brightness.



11

## MICROCONTROLLER DEVELOPMENT BOARD

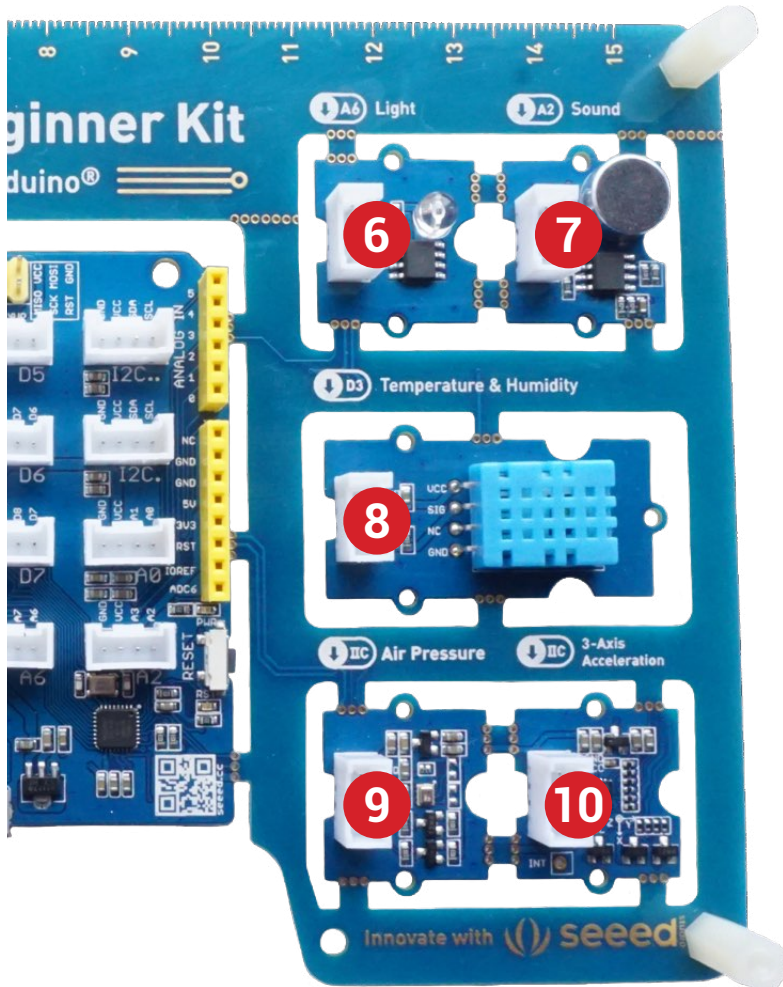
Based on the Arduino Uno and Seeeduino Lotus development board, this module is the brains of the board.

An ATmega328P microcontroller chip lies at the core, acting as small low-power computer that can be reprogrammed to create whatever device you wish.



The kit comes as an integrated PCB board with several useful input and output devices already connected and ready to go. No soldering, wiring or connecting of components means it's perfect for getting started with hardware!

Below is a quick summary of each component on the board and what they might be used for:



6

### LIGHT SENSOR

A photoresistor that can detect incident light intensity in the environment.

7

### SOUND SENSOR

A simple microphone that can detect the sound intensity in the environment.

8

### TEMPERATURE AND HUMIDITY SENSOR

Also known as a hygrometer. A pre-calibrated digital sensor that can measure environmental temperature and humidity. Will not work below 0°C.

9

### AIR PRESSURE SENSOR

Also known as a barometer. A high-precision digital sensor that can measure both air pressure and temperature. Can also be used to measure altitude.

10

### 3-AXIS ACCELERATION SENSOR

Also known as an accelerometer, which senses movement of the board. It can be used to measure orientation, tilting, movement or gestures.

The white plug sockets in the centre and yellow header sockets around the edges can be used to plug in additional components.

This module also has a reset button to reset your programme at any time, and a micro USB port, to connect the board to your computer.

A USB cable is provided in the right-hand compartment of the Grove box, and Grove cables (to connect components to the white sockets) are provided in the left-hand compartment of the Grove box.

# The Microcontroller

## What is a Microcontroller?

A microcontroller is a small low-power computer encapsulated in a tiny electronic chip. In contrast to a general purpose computer like a laptop or PC, microcontrollers are often designed to complete one task and run one specific programme. They are low cost and only require small amounts of power, so they are often used in simple electronic devices such as kitchen appliances, implantable medical devices and power tools.

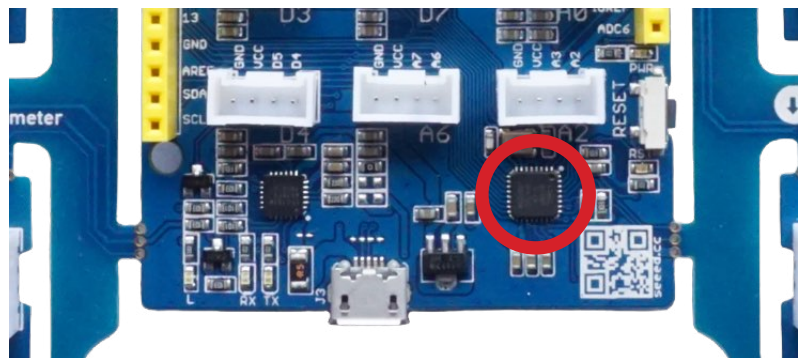


Like any other computer, a microcontroller has a Central Processing Unit (CPU), a 'long-term' memory (Electrically Programmable Read Only Memory, EPROM) for holding your programme, and a 'short-term' memory (Random Access Memory, RAM) for holding and accessing user data. It communicates with the outside world via a series of metal pins that can either send (output) or receive (input) information.

*The ATmega328P microcontroller used in the Arduino board*

A microcontroller development board, like the Grove Arduino board, houses a microcontroller chip on a small PCB board alongside some additional parts and connections making it easy for anyone to programme and connect components to a microcontroller.

Development boards are intended to be cheap and easily accessible, and are often used for developing prototypes and custom instruments. To get an idea of the wide range of projects that are possible to achieve using an Arduino development board, take a look at the project documentation platform Hackster at [www.hackster.io/Arduino](http://www.hackster.io/Arduino).

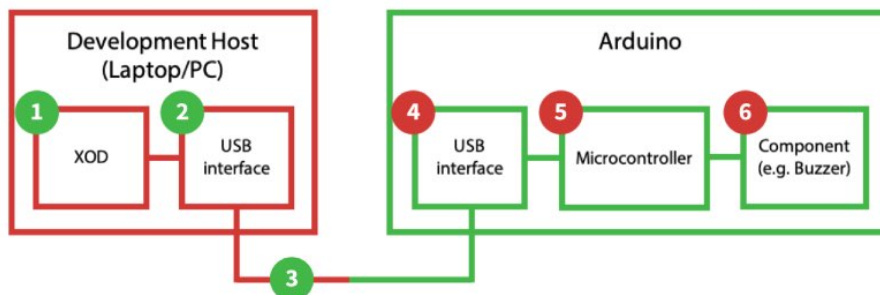


*Location of the ATmega328P microcontroller on the Grove board (red circle)*



## Controlling Your Arduino Board

In order to tell the Grove board what to do you will need to plug it into a computer. This is referred to as the development host, as it is where you will write and develop the programme you want to install. The diagram below explains how information is transferred from your computer to the board. Once the programme has been transferred, the board can be disconnected and will be able to run the desired programme independently of your computer, although it will need an alternative power supply. The board can be programmed to perform a multitude of different tasks depending on what components you want to use, and what programs you install.



*Workflow for programming your Arduino board*

- 1** Write your programme using XOD software
- 2** Upload your programme
- 3** Information is sent from the computer to the Grove board via a USB cable
- 4** Information is received by the Grove board
- 5** The programme is written to the EPROM (long-term) memory of the microcontroller chip. This allows the board to act as its own independent computer, carrying out the specific programme you have uploaded.
- 6** Information is sent to the onboard components via the microcontroller pins. The programme stored in the microcontroller's memory will tell the components what to do, for example, turn on the buzzer at a certain pitch.

# The Microcontroller

## Types of Communication

There are several different ways for the board to communicate with your components. These are known as communication protocols, and they are the different ways in which data can be transferred between devices. Which pin is connected to which device depends on what type of communication protocol is used, and that depends on the type of device. Below we describe the three types of communication that are used on the Grove board.

It is important to be aware of these different communication types, as they will determine the board's pin connects (see next page), as well as how to plug in any additional components you would like to add to your board.

### ANALOG

Analog sockets are used to connect analog input and output devices. They can transmit signals that are continuous (meaning they can have an infinite amount of values within a given range), unlike digital sockets which can only transmit signals in two states: on and off. Most sensors are analog sensors.

### DIGITAL

Digital sockets are also used to connect input and output devices. However, unlike analog devices, digital devices cannot take a range of values, they can only communicate by switching between two states: on and off. These digital sockets are used for most non-sensor components.

### I2C

For devices which deal with both inputs and outputs we need duplex communication protocols, which can transmit data in both directions. The duplex protocol used on the Grove board is I2C.

Inter-integrated circuit (I2C) pins provide a way to communicate with multiple devices at once. In this case, several devices are connected to the same pin, and each device is given a "name" (known as an address). Addresses are written as XXh, with XX being a two digit code of numbers and letters. For example 19h or 3Ch.

This covers three pin types used to connect the Grove board's inbuilt components. Arduino boards are also able to connect to devices using two other communication protocols, known as UART and SPI. We will not use these communication types in this guide, but if you would like to learn more you can find an excellent tutorial comparing I2C, UART and SPI protocols on the SparkFun website at [www.learn.sparkfun.com/tutorials/i2c](http://www.learn.sparkfun.com/tutorials/i2c).





## Pin Connections

Once your programme has been uploaded to the microcontroller chip, the chip needs to communicate with the board's components. Information can either be sent as an output from the microcontroller to the components, or received by the microcontroller as an input from a component.

Each device on the board is connected to both to the power source and to one or more of the pins (also known as 'ports') on the microcontroller chip. This allows the microcontroller to communicate with the components. It is important to know which component is connected to which pin, as we will need to use this information when we are programming.

The table below outlines which onboard devices are connected to which pins.

PIN	DEVICE
A0	Rotary Potentiometer
A2	Sound Sensor
A6	Light Sensor
D3	Temperature and Humidity Sensor
D4	LED
D5	Buzzer
D6	Button
I2C (19h)	Three-Axis Accelerator
I2C (77h)	Air Pressure Sensor
I2C (3Ch)	OLED Screen

# The XOD IDE

The XOD integrated development environment (IDE) is a free open-source software that allows you to programme microcontroller-based devices, such as Grove or Arduino boards, using visual 'nodes' rather than written code.

## 1 YOUR PATCH

The central part of your screen displays the XOD patch you currently have open. A patch is a space for you to create your programme. It is like a file in another programme, and can be used to create one small programme, or one section of a larger programme. You can create multiple patches and store them together in a project.

When you first open the XOD IDE you will see a project called 'welcome-to-xod'. This is a pre-installed tutorial from XOD. You can create a new project by navigating to 'File > New Project' in the menu bar.

## 2



The four buttons to the left of the Project Browser represent (from left-to-right):

### ADD PATCH

Lets you add a new patch to your project.

### ADD LIBRARY

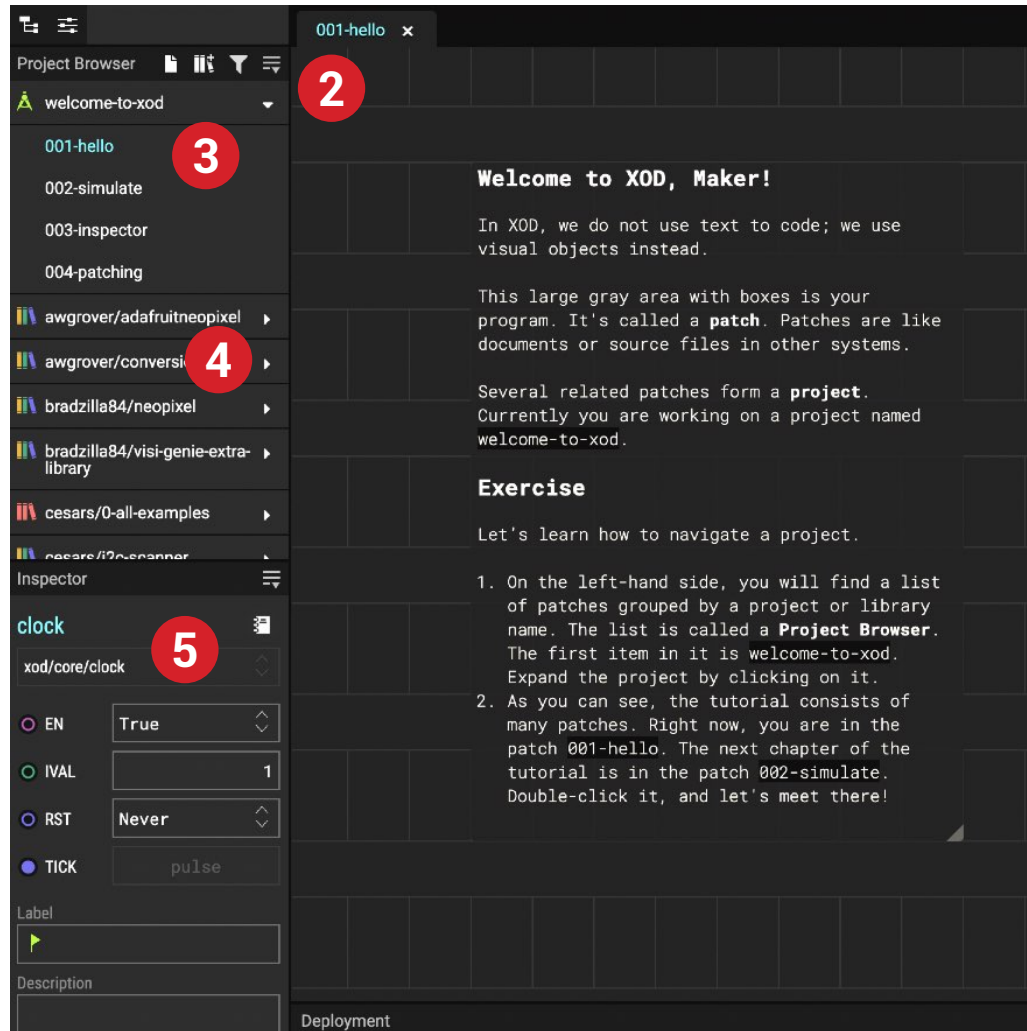
Lets you install new libraries. You can download libraries that other users have made to expand the number of nodes available.

### FILTER

Lets you filter what libraries and nodes you can see.

### PROJECT BROWSER MENU

Lets you minimise or move the Project Browser pane.



## 6 QUICK HELP

The quick help pane provides information about whatever node you have selected at the time. Click on a node to see information about what it does and what each of its inputs and outputs ('pins') do.

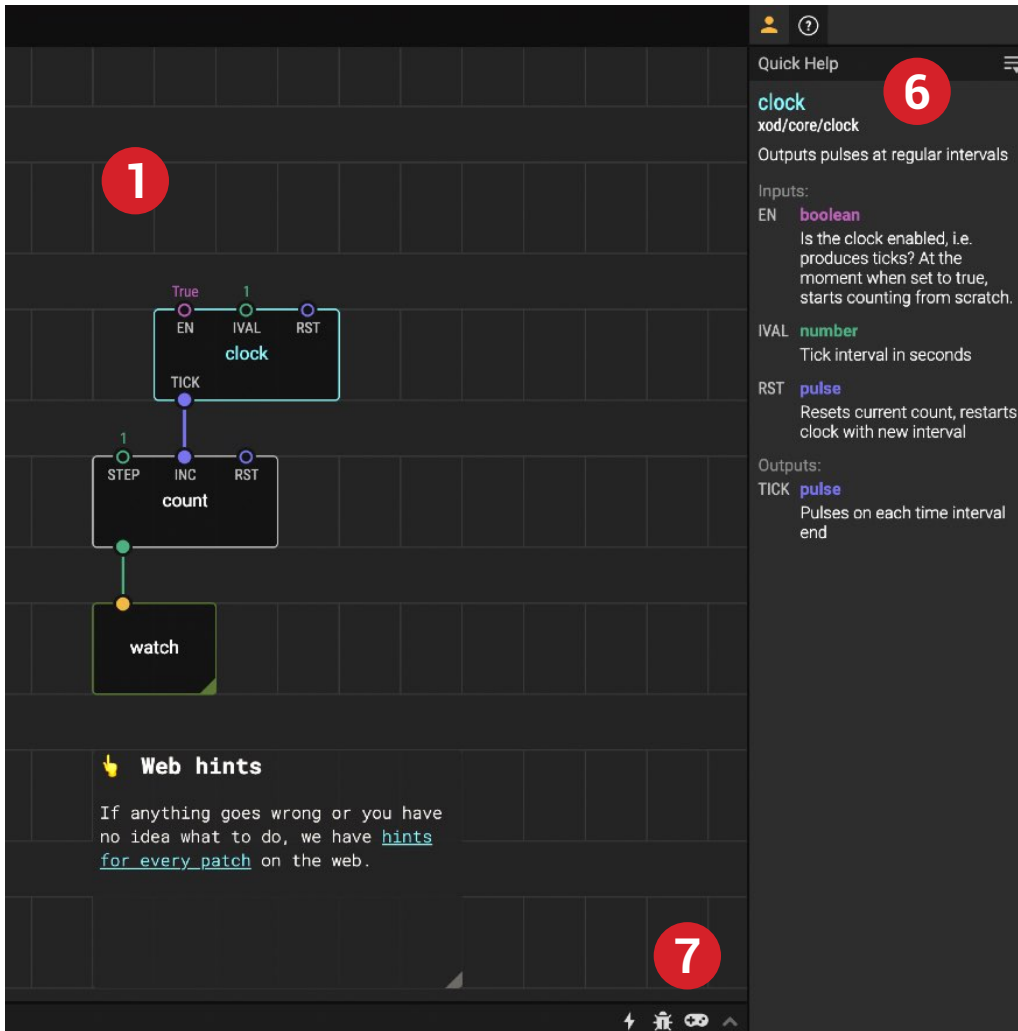
You can toggle this pane on and off using the question mark button at the top right of the screen.





Nodes can represent devices or functions, and by linking them together in different ways you can create a wide variety of different programmes. Programming visually like this can save some of the time and energy required to learn a new language and large amounts of syntax.

Below is a quick summary of the different parts of the XOD IDE, and what you'll see when you first load the software.



### 3 PROJECT BROWSER: PROJECT PATCHES

The top half of the Project Browser pane shows the project you have open. If you click the drop down arrow next to the project name you will be able to see all of the patches within your project.

### 4 PROJECT BROWSER: LIBRARIES

The bottom half of the Project Browser pane shows all of the libraries you have installed. When you first use XOD these will be limited to the basic XOD libraries (e.g. xod/bits, xod/core). Libraries with red icons have an error somewhere in their patches.

### 5 INSPECTOR

The Inspector pane shows information about the node or patch that you have selected at the time. This is where you will input information and change the properties of your nodes (e.g. you can tell the programme which of the microcontroller's ports your component is connected to).

This pane also contains the Label and Description boxes, which you can use to help document your programs

### 7



The four buttons at the bottom of the patch represent (from left-to-right):

#### UPLOAD TO ARDUINO

Upload your patch to the board.

#### UPLOAD AND DEBUG

Upload the patch, and watch what's

happening on your screen at the same time. Useful for testing programmes and debugging.

#### SIMULATE

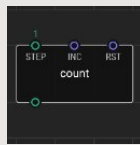
Simulate your programme without hardware. Useful for getting started.

#### TOGGLE DEPLOYMENT PANE

Toggle the Deployment pane on and off to see how the upload is working.

# The XOD IDE

## XOD Terminology



### Nodes

Nodes are the building blocks of a programme in XOD. Depicted as a black rectangle with white border, they will display their name in the middle, and have a number of small circular inputs and outputs on the top and bottom, known as pins. Nodes can represent any number of things, from hardware (like a LED or sensor) to mathematical or logical operations (like add, subtract, and, or, if etc.).



### Pins

In XOD, 'pin' refers to the inputs and outputs of a node, which are represented as small circles. Input pins are located on the top edge of a node, and output pins are located on the bottom edge of a node. Pins can have different data types (see next page) and are coloured accordingly. The name of a pin will appear below the circle, and the current value of a pin will appear above it.

Note: to avoid confusion with the 'pins' on the microcontroller, XOD refers to these as 'ports', i.e. what Grove refers to as 'pin A0' XOD refers to a 'port A0'.



### Links

Links are used to connect nodes. To create a link click on an output pin from one node, and then on an input pin from another node (or visa versa). Once a pin is linked, the circle will change to a solid colour. Pins of one data type can only be connected to certain other types (e.g. a number pin can be connected to another number pin, a string pin, or a boolean pin, but not a byte, port or pulse pin). XOD will not let you connect pin types that do not work together.

# XOD Data Types

XOD has built-in data types that represent the different kinds of values that can be used in a program. Here are some of the main data types in XOD:

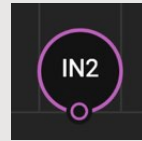
- **Number (num)**: This data type represents a floating-point number. It can be used for various purposes, such as storing sensor readings, controlling the position of a servo motor, or setting the duration of a delay.
- **Boolean (bool)**: This data type represents a binary value, either true or false. It is often used for conditional statements, like checking if a button is pressed or if a sensor value exceeds a certain threshold.
- **Pulse (pulse)**: This data type represents an event or a trigger. It is used to initiate an action, like starting a timer, toggling an output, or triggering a sequence of events. Pulses do not carry any additional information other than the fact that an event has occurred.
- **String (string)**: This data type represents a sequence of characters, like text or messages. It can be used to store and manipulate text data, display messages on an LCD screen, or send and receive data through serial communication.
- **Byte (byte)**: This data type represents an 8-bit value, ranging from 0 to 255. It is often used when working with low-level data, such as when communicating with a specific protocol or manipulating individual bits in a byte.
- **Port (port)**: This data type represents a pin on a microcontroller board. It is used to configure and control input/output pins, such as reading a digital signal, setting an output pin's state, or configuring a pin for PWM.
- **Color (color)**: This data type represents an RGB color value. It can be used to control the color of RGB LEDs, create color gradients, or manipulate colors in graphical displays.

Pins in XOD can take a number of different data types depending on what they represent. Each data type is represented by a distinct colour. Below is a description of the six main data types in XOD. Custom data types are also available, but we will not discuss these here. A comprehensive explanation of each XOD data type and how they interact can be found at [www.xod.io/docs/guide/data-types](http://www.xod.io/docs/guide/data-types) and [www.xod.io/docs/reference/data-types](http://www.xod.io/docs/reference/data-types) respectively.



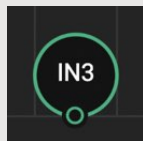
## Pulse

Pulse pins are like triggers. They don't represent any specific type of data, but they are used to signify when something has happened, or to trigger something to happen at a specific time.



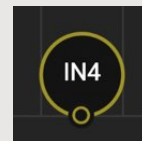
## Boolean

Boolean pins represent logical information, i.e. they can only be in two states: True (on) or False (off). They can be used like a switch, and are often used when working with logic nodes such as 'and' 'or' etc.



## Number

Number pins are very simple: they represent numbers. Numbers can be integers or fractions in the range  $\pm 16$  millions, and can display up to 6 significant digits.



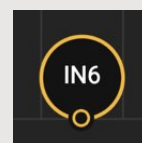
## Byte

Byte pins represent bytes, a fundamental data type in computing. They can be written in a variety of ways, but we will use hexadecimals, which contains two digits (0-9,A-F) followed by h-suffix (e.g. 3Ch).



## Port

Port pins represent the different 'ports' (microcontroller pins) on the board, i.e. A0-A6 and D0-D13. We will use these pins to tell hardware nodes which pin/port the hardware is connected to.



## String

Strings pins represent strings of text. They are used to input text or read out text. Usually this is text input by or to be read by the user, e.g. text to be displayed on a screen.





# Lesson 2: Getting Started



**Setting up your board**



**Testing your board**



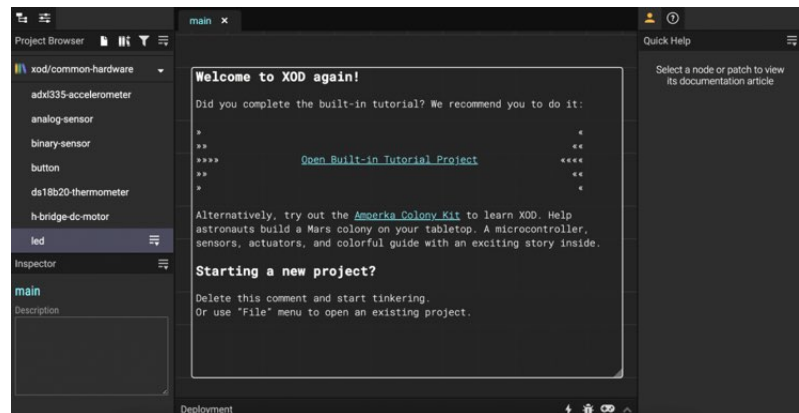
**Input and Output devices**

# Getting started

This chapter is all about how to get started with no-code programming and using your board.

Beginning with 'Setting up your Board', you will learn how to set up your computer, including how to download the XOD IDE, USB drivers and Biomaker tutorial files.

This is followed by two tasks: 'Testing your Board' and 'Input and Output Devices'. In Task 1 you will use the XOD IDE to test your connection and programme the simple LED on your board. In Task 2 you will learn how to build a simple device using the button and buzzer modules on your board, as well as expanding your knowledge of how to use XOD.



The XOD welcome screen

## OBJECTIVES

By the end of this chapter you should be able to:

- Prepare your Biomaker starter kit by downloading the relevant software and drivers, plugging in your board, and opening the XOD IDE.
- Name the different sections of the XOD IDE and understand what they are used for.
- Apply your knowledge of the XOD IDE to perform the following simple tasks: add a node, change pin settings, connect nodes, add a library.
- Use the XOD IDE to upload programmes to your board.
- Use the XOD IDE to clear programmes from your board.
- Use three of the inbuilt components on the Grove board: the LED, the buzzer and the button.
- Understand how to troubleshoot your programme and find additional help.
- Understand how input and output devices can be used together to build simple devices.

# Setting up your board

## Downloads

### XOD

To download the free XOD software, simply visit [www.xod.io](http://www.xod.io) and download the desktop IDE from the XOD homepage. You will need to download the correct IDE for your operating system (Windows, MacOS, Linux etc.).

Note that a browser-based IDE is also available, but does not support hardware, so is not suitable for use with this guide.

### USB DRIVER

For your computer to communicate with your Arduino board it will need to have the correct driver installed, in this case, a CP210 driver. Most operating systems will already have the correct drivers installed, including: Windows 7 and 10, Mac OSX v10.10.5 (Yosemite) to v10.15.5 (Catalina), Linux and Ubuntu v18.04.2, 64-bit.

If you are using one of the above systems we suggest ignoring this step and continuing with the guide.

If you are using a different operating system, or are having trouble connecting with your board, you may need to download a CP210 driver. Downloads for most common operating systems are available from Silabs at: [www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers](http://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers).

### TUTORIALS

To accompany this guide, the Biomaker team has created a XOD tutorial file. This file will allow you to work through the tasks in this guide within the XOD environment.

You can choose to work through the tasks by using the step-by-step instructions in this book, by using the XOD tutorial file, or by using a combination of both.

If you chose to use the XOD tutorial file, we advise that you take the time to read though the introduction and information at the start of each chapter in this guide, otherwise you may miss out on useful information.

You can download the XOD tutorial file on the Biomaker website at: [www.biomaker.org/nocode-programming-for-biology-handbook](http://www.biomaker.org/nocode-programming-for-biology-handbook).



# Testing your board

## Task 1: Requirements

- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (LED and Button modules)
- USB-A to micro USB cable

Now that you've downloaded the software you're all set and ready to get stuck in!

This task will walk you through how to connect your board to the computer and upload your first programme using XOD.

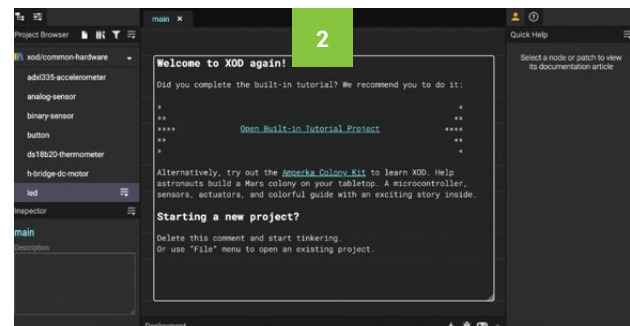
We'll be using the LED light in the top left corner of the board to test our connection.

You may notice that when you first plug in the board the OLED screen in the middle-left turns on. This is part of the inbuilt demo mode on the board. You can learn more about this in the Grove User Manual.



### PLUG IN YOUR BOARD

Use the USB cable provided to plug your board into the computer. You can find this cable in the right-hand compartment of the Grove box. Plug the micro USB end into the socket at the bottom of the central section of the board, and the standard USB (USB-A) end into your computer.

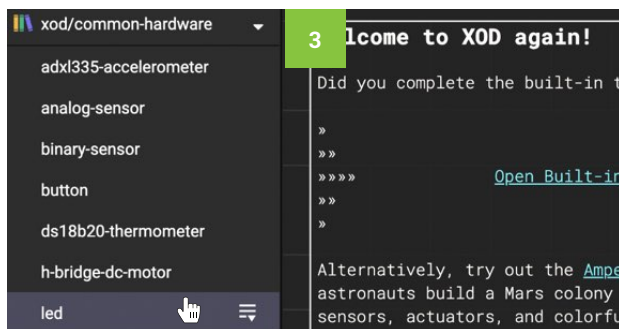


### OPEN THE XOD IDE

Open up the XOD software on your computer.

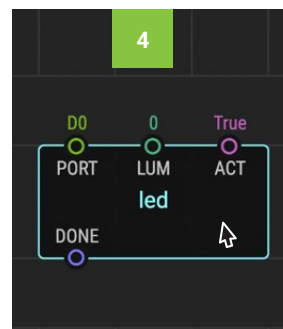
If you are using the tutorial file provided, open this file in XOD. You can follow the instructions in patches *tuto101-tuto114* to complete this task.





### ADD AN LED NODE

Using the Project Browser on the top left, find the **xod/common-hardware** library, click the dropdown menu and find the node called **"led"**. Click on this node and drag it into the patch. This is one of several ways to add a new node. See the Adding Nodes to your Patch box for more information on alternative methods.



### SELECT THE NODE

Click on the **led** node. The outer edge will turn blue and more information will appear in the Inspector pane on the bottom left.

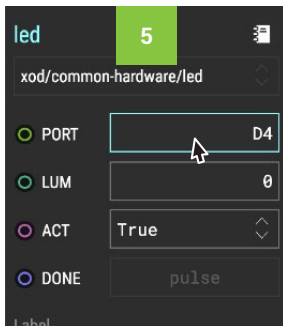


## Adding Nodes to your Patch

There are several different ways to add a node to your XOD patch, and which one you use is completely up to you!

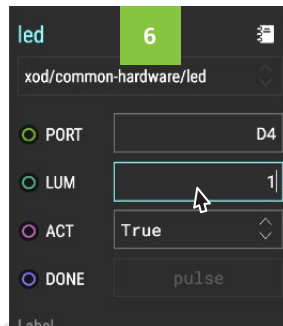
1. **DRAG FROM LIBRARIES**  
As described above. If you know the library the node is in, you can find the library in the Project Browser, click the dropdown menu, click on the node, drag it into the patch and release.
2. **DOUBLE CLICK ON THE PATCH**  
If you know the name of the node you want, or want to search for a node you can use the search bar. Double click anywhere in your patch and the search bar will appear. Start typing the name of the node and options will appear. Click on the correct node and it will insert into your patch.
3. **KEYBOARD SHORTCUT**  
Similar to double-clicking the patch. Click anywhere on the patch and press 'i' on your keyboard. This will bring up the same search bar as above.
4. **MENU BAR**  
This is a third way to bring up the search bar. Select 'Edit > Insert Node...' from the menu bar.

# Testing your board



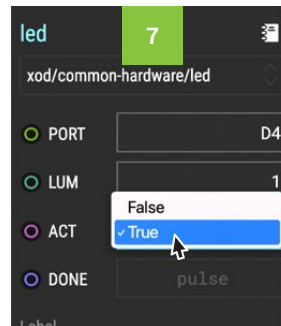
## SET PORT PIN

The LED on the board is connected to port D4, so click on the text box next to PORT and set this to D4 by typing 'D4'.



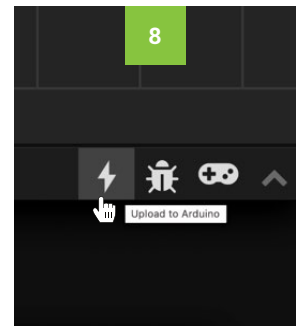
## SET LUM PIN

LUM stands for luminance, i.e. how bright the LED is on a scale of 0-1. Set this to 1 (brightest level) by typing '1'.



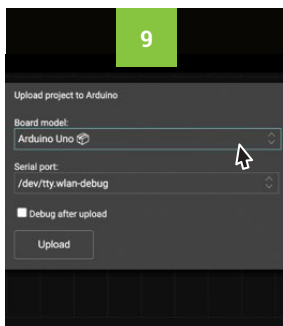
## SET ACT PIN

ACT is a boolean pin that can only be true or false. Use the dropdown to set this to 'True'. This makes sure the LED responds.



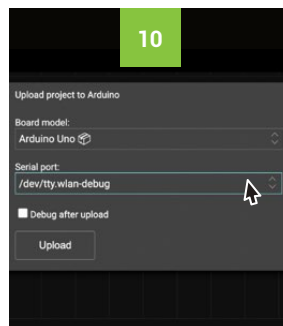
## UPLOAD

Click on the small lightning icon in the bottom right, or select 'Deploy > Upload to Arduino...' from the menu bar.



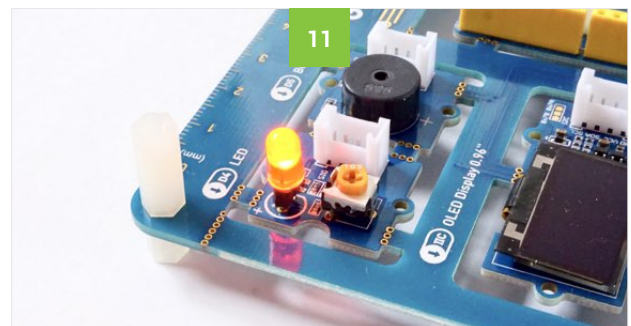
## SET BOARD MODEL

Use the dropdown menu to select 'Arduino Uno' or 'Arduino/Genuino Uno'.



## SET SERIAL PORT

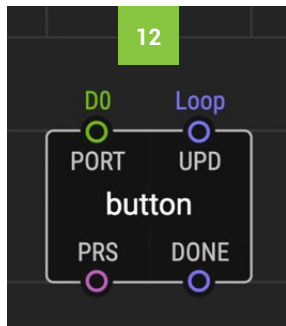
Use the dropdown menu to select the option that ends in '(Silicon Labs)'.



## WATCH YOUR LED!

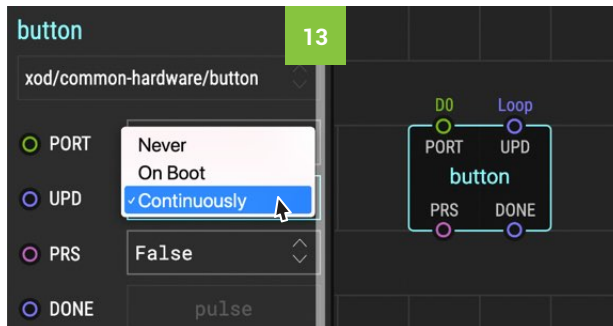
Click upload and watch the LED on your board. It should light up!

If not, don't worry! Take a look at the Troubleshooting box on the next page (p25).



#### ADD A BUTTON NODE

Now let's add another node. Using one of the ways described on p21, add a *button* node from the **xod/common-hardware** library.



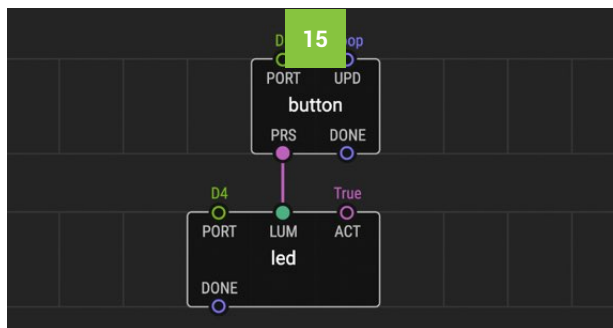
#### SET BUTTON PINS: UPD PIN

The UPD pin specifies how often the programme updates. This can be set to 'Never', 'On Boot (Boot)', or 'Continuously (Loop)'. Alternatively another node can be connected to this pin and used to determine how often it updates. Make sure this is set to 'Continuously (Loop)', so that whenever we press the button it is read instantly.



#### SET BUTTON PINS: PORT PIN

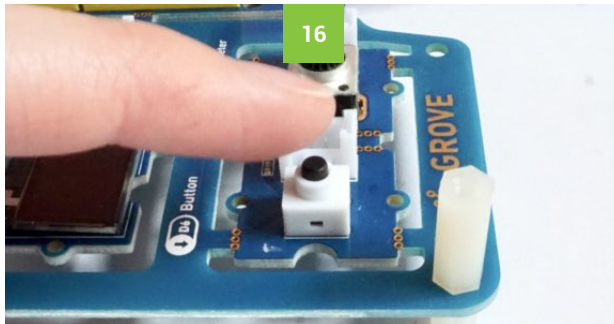
As with the LED node, PORT specifies which port the button is connected to. Set this to 'D6'.



#### CONNECT THE NODES

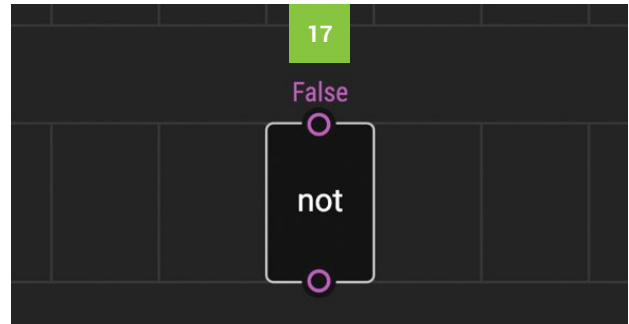
We want the LED to turn on whenever we press the button, so we need to connect the **button** output pin PRS (press) to the **led** input pin LUM. Do this by clicking on the PRS pin and then on the LUM pin. Now when you click on the *led* node you will not be able to set the LUM pin, because it's value is determined by **button** node.

# Testing your board



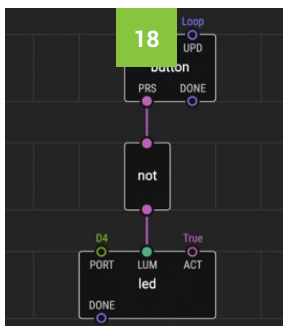
## UPLOAD AND TEST

Upload the patch and see what happens. You will notice that the programme is backwards. The LED is on and turns off when you press it. This is because the board's buttons are set to be on by default, and turn off when pressed. We can fix this programme with a logic node.



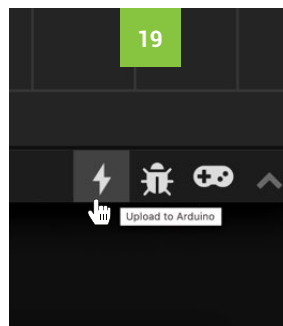
## ADD A NOT NODE

To invert the signal from the button we can use a different type of node that represent a logic function, rather than a piece of hardware. Insert a **not** node from the **xod/core** library.



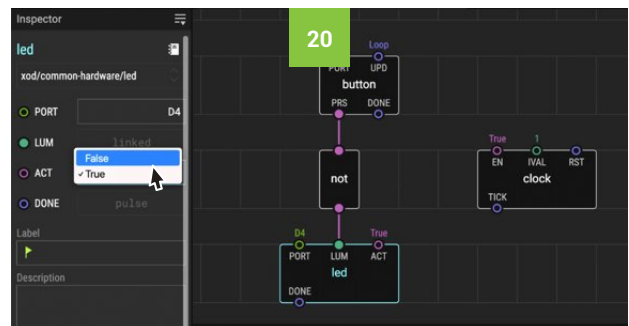
## REWIRE THE PATCH

Delete the link between the **button** and **led**. Connect PRS to the **not** input pin, and the **not** output pin to LUM.



## UPLOAD AND TEST

Now try uploading your programme again. This time it should work as planned.

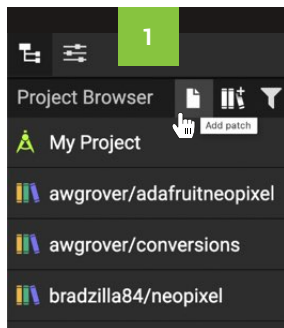


## EXPERIMENT!

Congrats! you've now made a simple programme that uses an input (the button) and an output (the led) to affect change. Why not try experimenting with this patch? Play around with some pins. E.g. change the led ACT pin, or link a clock node to the button UPD pin and see what happens. See what you can achieve!

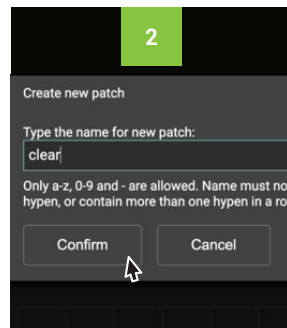
# Clearing the Board

Once a programme is loaded onto your board it will remain there and restart whenever you turn on the board. Each time you upload a new programme it will write over the previous programme. You don't need to clear the board before you upload a new programme, but if you wish to reset your board you can do this manually by uploading a blank patch in XOD.



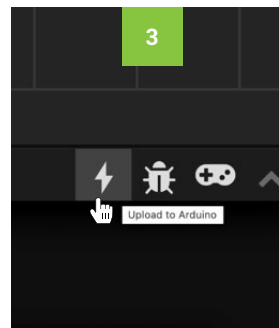
## MAKE A NEW PATCH

Add a new patch by clicking the 'Add patch' button in the project browser or selecting 'File > New Patch...' from the menu.



## NAME THE PATCH

Type a name for your new patch, e.g. 'clear' and click confirm or press the Enter key.



## UPLOAD

Upload the empty patch by clicking the upload button as before. This will turn off the LED and clear the board.



# Troubleshooting

If your LED doesn't light up straight away there a few quick things to check:

**1. IS THE BOARD PLUGGED IN CORRECTLY?**

If your board is plugged in correctly the power light on the right side of the Seeeduino module should light up. If not, make sure that the USB cable is plugged in fully.

**2. HAVE YOU SET YOUR NODE PARAMETERS CORRECTLY?**

Setting the wrong parameters is a common mistake. In this case you should make sure the pins are set as follows:  
PORT = D4   LUM = 1   ACT = True

**3. HAVE YOU UPLOADED USING THE RIGHT BOARD MODEL AND SERIAL PORT?**

After clicking the upload button you should make sure that you have the correct board model and serial port selected. Use the dropdown menus to select 'Arduino Uno'/'Arduino/Genuino Uno' and 'dev/tty.usbserial-0001 (Silicon Labs)'.

**4. DO YOU NEED TO INSTALL A USB DRIVER?**

If XOD is not recognising your board, you may need to install a CP210 USB driver (see page 15).

Still need help? XOD provides *watch* and *tweak* nodes which are useful for troubleshooting and debugging your programme. Read more about them on page 27 of this guide, or take a look at XOD's guide to debugging at [www.xod.io/docs/guide/debugging](http://www.xod.io/docs/guide/debugging). For further help you can always contact the Biomaker team at [coordinator@synbio.cam.ac.uk](mailto:coordinator@synbio.cam.ac.uk).

# Input and Output devices

## Task 2: Requirements

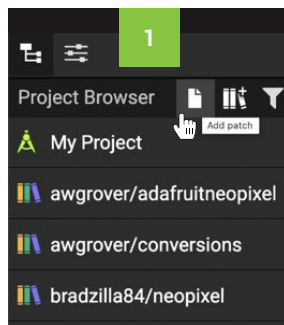
- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (buzzer , button and rotary potentiometer modules)
- USB-A to micro USB cable

Great! You now understand the basic principles of using XOD, and can programme your board to control one of the onboard devices: the LED.

Like the button and the LED, most devices you will use can be grouped into two general categories: inputs and outputs.

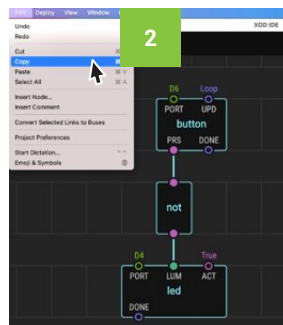
Understanding how to control different inputs and outputs, and how to combine them together is key to making useful devices.

In this task we will build on our knowledge to add two new devices to our belt. The buzzer and the rotary potentiometer (also known as a knob sensor).



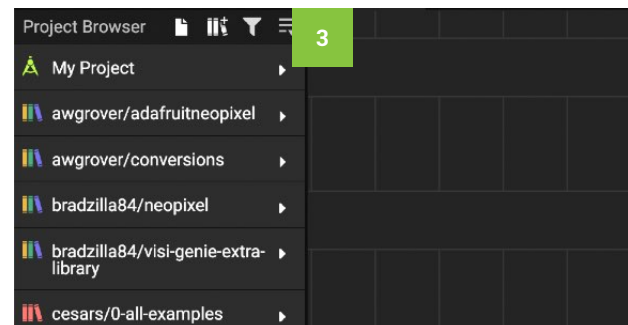
### MAKE A NEW PATCH

Follow the instructions in 'Clearing the Board' to open and name a new patch. (If you are using the tutorial file, move on to tuto201.)



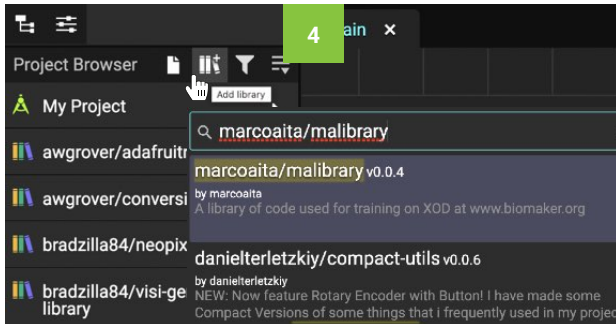
### COPY YOUR PATCH

Drag a box around all three nodes in the last patch. Copy and paste into your new patch. This will save us some work!



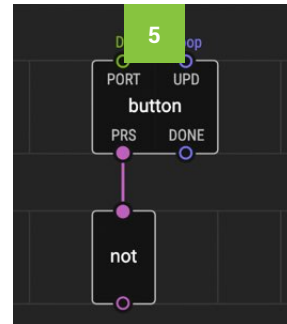
### FINDING A BUZZER NODE

There is no preinstalled node to represent the buzzer, but several have been created by members of the XOD community. We will use one from the library called **marcoaita/malibrary**.



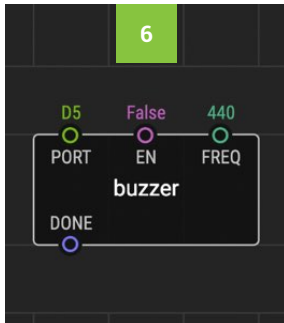
#### ADD A LIBRARY

Add this library by clicking on the 'Add library' button (next to the 'New patch' button at the top of the Project Browser) or by navigating to 'File > Add Library...'. Type the full name of the library, and when it appears, click on it to install. If you get an error message asking to install dependencies, accept this.



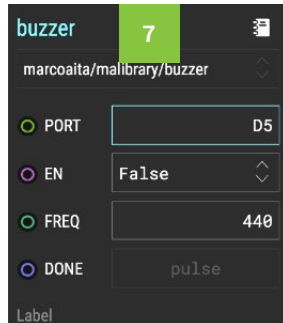
#### DELETE LED NODE

This time we want to use the **buzzer** as an output instead of the LED. Click on the led node and delete it.



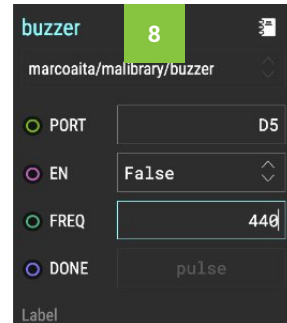
#### ADD A BUZZER NODE

Now that the library is installed you can search for the **marcoaita/malibrary/buzzer** node and add it as usual.



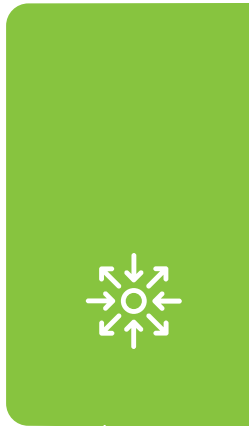
#### SET BUZZER PINS: PORT

The buzzer is connected to port D5, so make sure the PORT pin is set to 'D5'

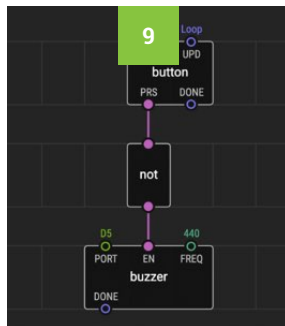


#### SET BUZZER PINS: FREQ

The FREQ pin sets the frequency and pitch of the buzzer. You can leave this as 440, or change it too see what happens.

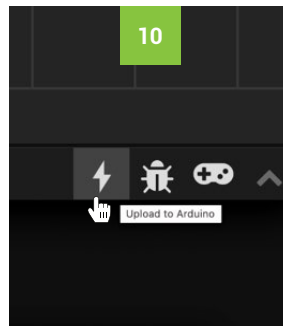


# Input and Output devices



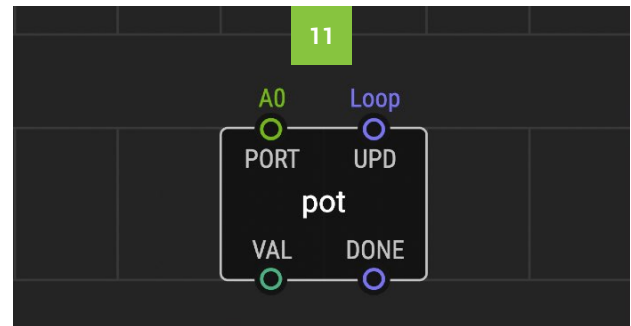
## RECONNECT THE NODES

Connect the **not** output pin to the buzzer EN (enabled) pin. This will 'enable' the buzzer when the button is pressed.



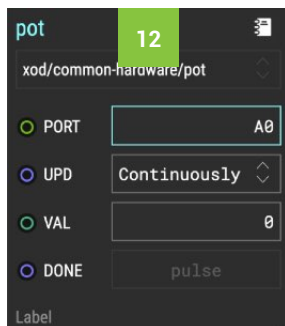
## UPLOAD AND TEST

Now try uploading your programme. It should work similarly to the LED patch, i.e. the buzzer turns on when you press the button.



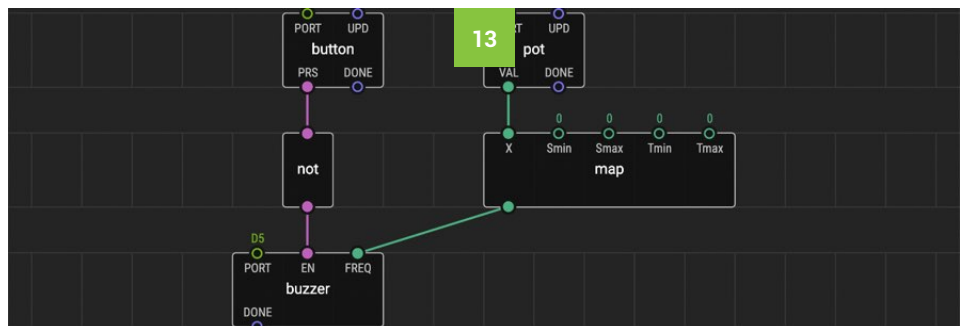
## ADD A POT NODE

Now let's add a second input. We can use the inbuilt rotary potentiometer (knob) to adjust the frequency of the buzzer sound. To represent the potentiometer we can use the **pot** node from the **xod/common-hardware** library. Add a *pot* node to the patch.



## SET POT PINS

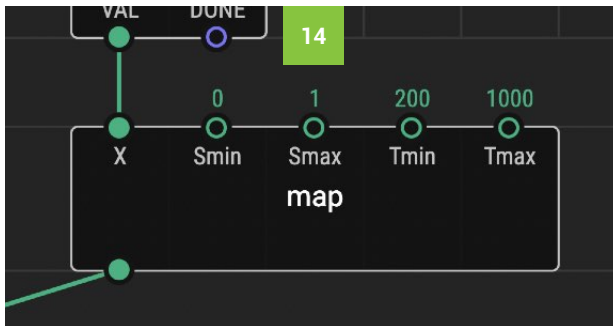
The potentiometer is connected to port A0, so set PORT to A0, set UPD to 'Continuously'.



## MAPPING VALUES: ADD AND CONNECT A MAP NODE

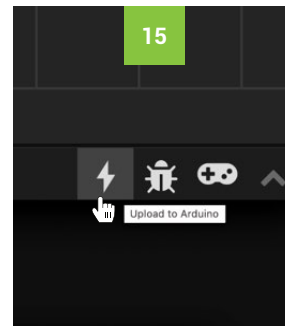
We could connect the *pot* output VAL (value) pin straight to the FREQ input pin. However, this wouldn't work well, as the VAL output ranges between 0 and 1, and frequencies emitted by the buzzer are much higher. To get around this, we can add a **map** node. This will 'map' your input range to a new output range, so we can change the 0-1 scale of the potentiometer to a larger scale that the buzzer can use. Add a *map* node from **xod/math**. Connect the **pot** VAL pin to the **map** X pin and the **map** output pin to the **buzzer** FREQ pin.





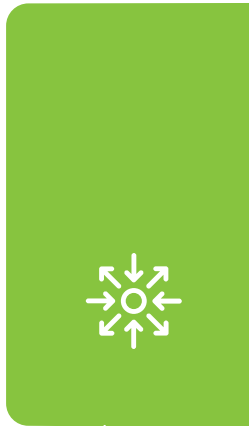
#### SET MAP PINS

Smin and Smax set the source range, whilst Tmin and Tmax set the target range. Set Smin to '0' and Smax to '1'. Set Tmin to '200' and Tmax to '1000'.



#### UPLOAD AND TEST

Now try uploading your programme. Use the button to turn the buzzer on and off, and the potentiometer to set the frequency.



## Building Complex Devices

When building biological devices, you will need to combine a variety of inputs and outputs to create a functioning programme and device. You will often receive inputs, e.g. from an on/off button or sensor, and then use these inputs to create the desired output, e.g. displaying a reading, sending data to a computer or moving a motor.

The button, buzzer and potentiometer circuit used here is a very simple example, but the principle applies in more complex systems too. Using XOD allows you to visualise this information flow from input to output, which can be helpful and sometimes more intuitive than traditional text-based coding.

In the next lesson we'll be getting a better understanding of what is possible in XOD by exploring a variety of useful nodes and processes using a range of the board's inbuilt devices.

When you are ready to explore beyond the starter kit's capabilities, the Resources tab of the Biomaker website explores a variety of common input and output devices which are useful for building biological devices.



# Lesson 3: Explore XOD



**Tweak and Watch nodes**



**Flip, Clock and Count nodes**



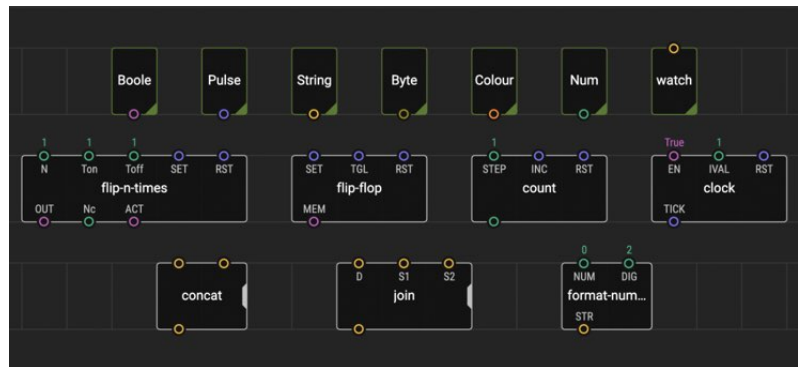
**Concat, Join and Format-Number nodes**

# Explore XOD

This chapter will explore some of the most useful nodes XOD has to offer. These nodes are used very commonly when building simple instruments, and will give you a good base to start from when exploring more complex devices.

The sections in this chapter are split into three tasks. First, following on from Task 2 in the previous chapter, Task 3 explores 'Tweak and Watch Nodes', which are useful for simulating and troubleshooting. Second, Task 4 examines 'Flip, Clock and Count Nodes' which are useful for ensuring correct timing of programmes. Finally, Task 5 looks at 'Concat, Join and Format-Number Nodes' which are useful for using and formatting text in XOD.

This chapter also encourages you to experiment with your use of XOD. It provides suggestions for how to expand the tasks, and encourages you to start thinking about the different ways in which you can achieve a desired outcome using no-code programming.



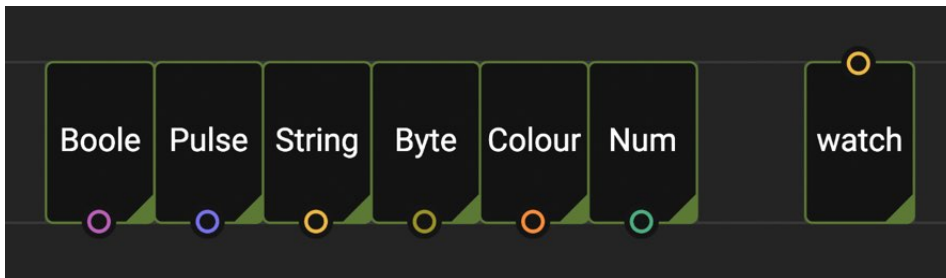
Useful XOD nodes

## OBJECTIVES

By the end of this chapter you should be able to:

- Explain the functions of the following XOD nodes: *tweak*, *watch*, **flip-n-times**, **flip-flop**, **clock**, **count**, **concat**, **join**, **format-number**.
- Apply your knowledge of these nodes to start building simple programmes.
- Use the XOD IDE to create and save a new project.
- Use the XOD IDE to 'upload and debug' programmes, allowing you to watch and edit your programme live.
- Use two more of the inbuilt components on the Grove board: the temperature and humidity sensor and the air pressure sensor.
- Build and compare different versions of a programme to achieve different functions and outcomes.
- Experiment with the programmes you have built by changing parameters and exploring new nodes
- Understand where to find more information about the basic nodes available in XOD

# Tweak and Watch nodes



## TWEAK NODES

The **tweak** nodes provided in XOD are a great way to edit your programmes whilst they are running. They are used in conjunction with the 'Simulate' and 'Upload and Debug' functions of XOD to edit programmes in real time, meaning that you don't have to reload the programme each time you want to make a small change, like altering the value of a pin.

There are multiple *tweak* nodes available depending on what type of pin you would like to change, and you will need to use the matching **tweak** node for the pin type, i.e. **tweak-boolean**, **tweak-pulse**, **tweak-byte**, **tweak-colour**, and **tweak-number**. There are also several **tweak-string** nodes depending on the size of string you want to input, e.g. **tweak-string-16** allows you to input a string of up to 16 characters, whilst **tweak-string-128** allows you to input up to 128 characters.

To use *tweak* nodes you will need to use the 'Upload and Debug' button rather than the 'Upload to Arduino' button, as this opens XOD's 'Debugger' function, which lets you live edit nodes. To edit a *tweak* node in the Debugger, click on the node and you will now be able to make changes in the Inspector whilst the programme is running.

## WATCH NODE

The **watch** node is the opposite of a **tweak** node. Instead of letting you input data, it lets you view output data whilst the programme is running. Connecting a **watch** node to any output pin lets you view the current value of that pin. This is useful for being able to visualise what the programme is doing, and where any problems are occurring.

A **watch** node can be connected to any output except a pulse output. To visualise the output from a pulse pin, you can use a **count** node in combination with a **watch** node. This is discussed further below.

Like **tweak** nodes, *watch* nodes need to be used in conjunction with the Debugger function. When the Debugger opens, the **watch** node will turn green and display the last value received from the connected pin.



# Tweak and Watch nodes

## Task 3: Requirements

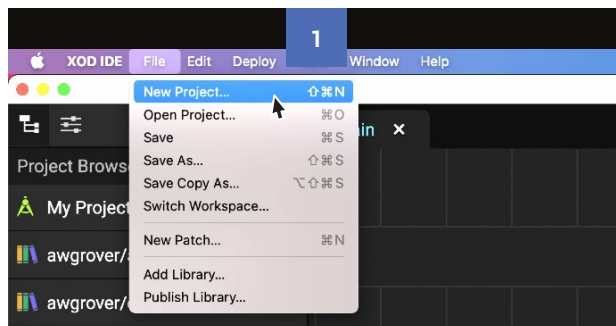
- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (temperature and humidity sensor module)
- USB-A or USB-C to micro USB cable

In this task we'll look at how we can use **tweak** and **watch** nodes to take readings from another of the inbuilt devices: the temperature and humidity sensor.

We'll be using a **tweak-pulse** node to act as a button and take a reading whenever we press (or 'tweak') it, and watch nodes to display our readings on the computer screen.

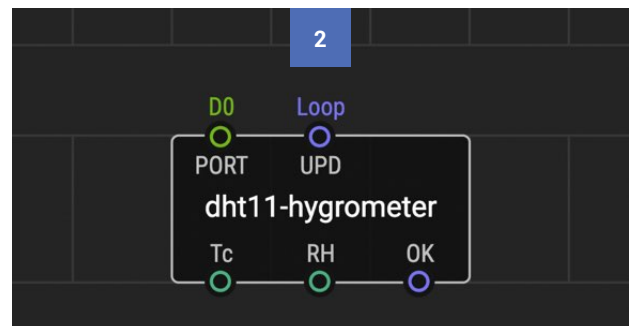
We'll also be using the 'Simulate/Debug' mode in XOD, which lets us watch and make changes while the code is running.

This is a great example of how **tweak** and **watch** nodes can be used to quickly and easily test a patch. They are very useful for testing and debugging patches, so you should try to get used to using them as you build.



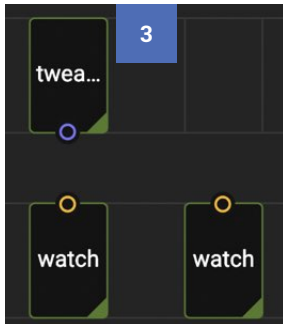
### MAKE A NEW PROJECT

A new chapter deserves a clean slate, so let's look at how we start a new project. First, save your old project (you can't have two projects open at once). Then navigate to 'File > New Project...' in the menu bar. Finally, give your project a name, and you can get started. (If you are using the XOD tutorial file, move on to tuto301).



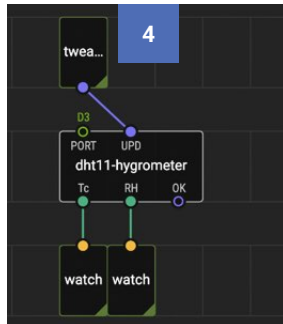
### ADD AND SET HYGROMETER NODE

The onboard temperature and humidity sensor is technically known as a DHT11 hygrometer. There is a preinstalled XOD node for this device called **dht11-hygrometer**. Add this node to your patch from the **xod-dev/dht** library. Set the port pin to 'D3'



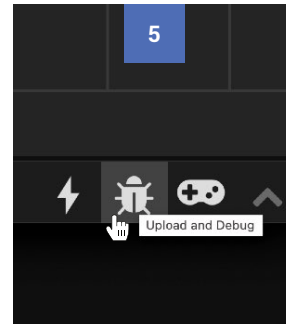
### ADD TWEAK AND WATCH NODES

Add a **tweak-pulse** and two **watch** nodes from the **xod/debug** library.



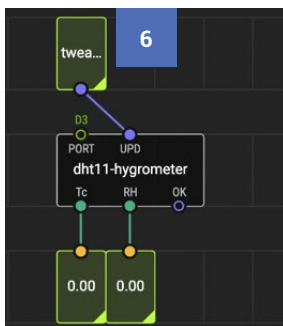
### CONNECT THE NODES

Connect the **tweak-pulse** node to the UPD pin and a watch node to each of the pins Tc (temperature °C) and RH (relative humidity).



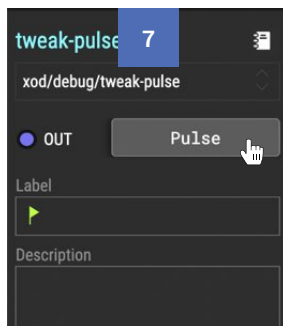
### UPLOAD AND DEBUG

Click the 'Upload and Debug' button (beetle icon) or use the upload button and tick the box labelled 'Debug after upload'.



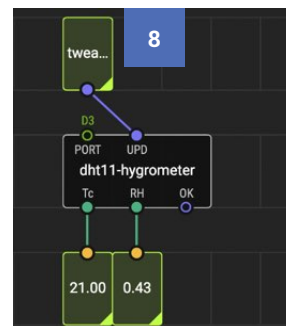
### WATCH

Look at the **watch** node. It won't display a reading yet because we have set the UPD pin to only update when we tweak it.



### 'TWEAK' THE NODE

Whilst it is still running, click on the **tweak** node. In the Inspector pane is a button next to OUT that says 'pulse'. Click it.

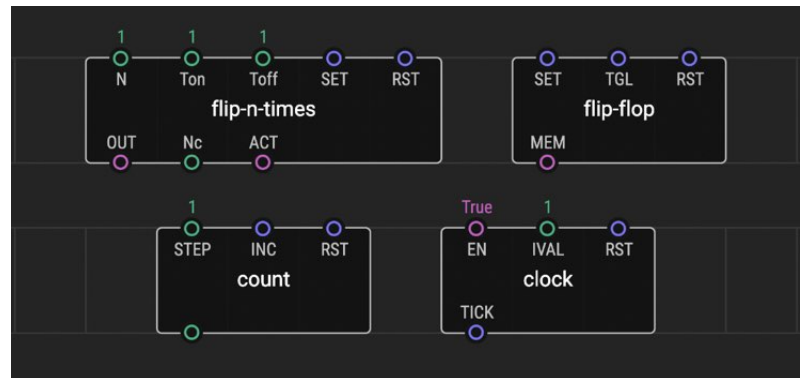


### WATCH AGAIN

Look at the **watch** nodes again. This time they should display the current temperature and humidity.



# Flip, Clock and Count nodes



## FLIP NODES

Flip nodes are boolean logic nodes that switch (or 'flip') between two states: 'True' and 'False'. There are two useful flip nodes in XOD: **flip-n-times** and **flip-flop**.

The first node, **flip-n-times**, will switch between 'True' and 'False' a set number of times (N). You can determine the time spent in each state using the Ton and Toff pins, and the whole sequence will be initiated by a pulse to the SET pin. This node is useful for creating sequences and patterns.

The second node, **flip-flop**, will switch between 'True' and 'False' states each time the toggle (TGL) pin receives a pulse. This is a particularly useful node which can act as a toggle or switch in many situations.

## CLOCK NODE

Like the flip nodes, the **clock** node is also useful for controlling the timing of your programmes. However, instead of giving a boolean 'True'/'False' output, the **clock** node sends pulses at a specific time interval. This node creates a regular 'ticking' of pulses, which can be used to control your programme.

## COUNT NODE

The **count** node is complimentary to the **flip** and **clock** nodes, and acts as a measure of how many times a pulse or boolean 'True' signal has been sent. This is useful for keeping track of your programme and it's progress. The **clock** node is also very useful, when used in conjunction with a **watch** node, to visualise the output from a pulse pin (see [Task 4, Steps 15-16](#) for an example of this use).

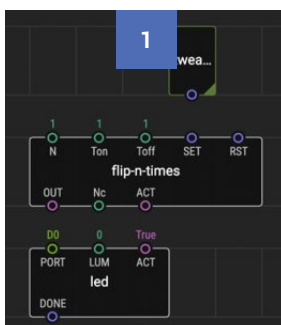


## Task 4: Requirements

- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (LED module)
- USB-A to micro USB cable

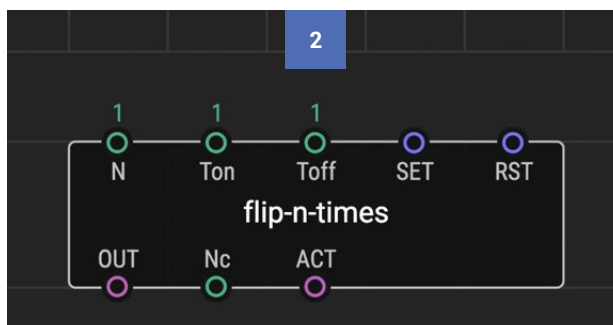
In this task we'll be experimenting with flip, **clock** and **count** nodes to control the behaviour of the inbuilt LED, making it flash.

The flip and **clock** nodes can be useful for modifying and timing the behaviour of nodes, whilst the **count** node can be useful for monitoring these behaviours. In the context of biological devices, these nodes are very useful for fine-tuning devices and for building larger programmes.



### NEW PATCH AND ADD NODES

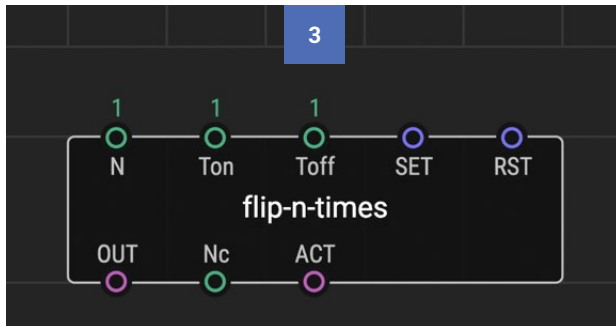
Open a patch (or go to [tuto401](#)). Add these nodes: **tweak-pulse** (`xod/debug`), **flip-n-times** (`xod/core`) and **led** (`xod/common-hardware`).



### FLIP-N-TIMES NODE INPUTS

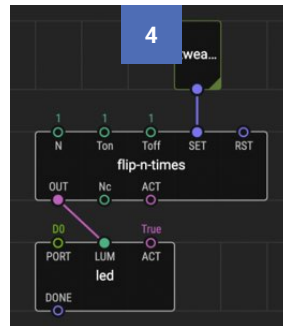
This node has 5 input pins: SET, RST (reset), N (number), Ton (time on) and Toff (time off). This node defines a sequence that will switch between true and false N number of times. Ton and Toff define the duration of each on and off state. A pulse to SET will start the sequence, and a pulse to RST will reset the node.

# Flip, Clock and Count nodes



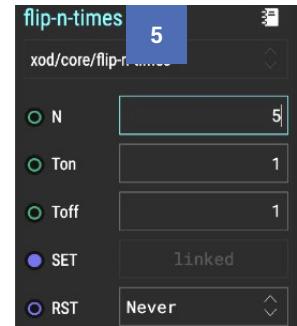
## FLIP-N-TIMES NODE OUTPUTS

The **flip-n-times** node has three outputs. OUT reads the current state of the node (true/false). Nc reads the number of times cycled. ACT reads whether the sequence is currently running or not. If you'd like to get a better idea of how these outputs work, you can always add **watch** nodes, to help see what's going on.



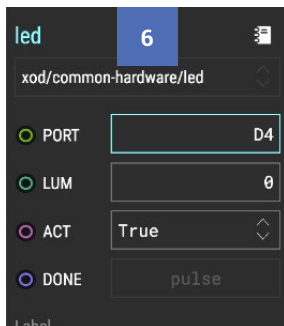
## CONNECT THE NODES

Connect the **tweak-pulse** node to the **flip-n-times** SET pin, and **flip-n-times** OUT pin to the **led** LUM pin.



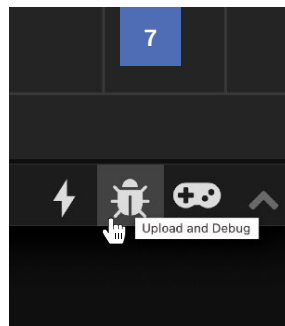
## SET FLIP-N-TIMES PINS

Set N to '5', Ton and Toff to '1', and RST to 'Never'. You can also add a **tweak-pulse** node to the RST pin to test how it works.



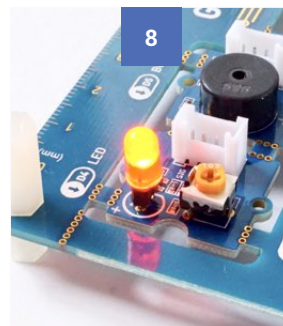
## SET LED PINS

As in [Task 1 Steps 5-7](#) (p22), set the PORT pin to 'D4' and the ACT pin to 'True'.



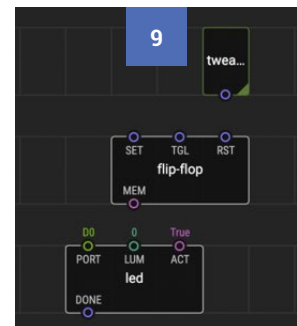
## UPLOAD AND DEBUG

Upload and run the programme in debug mode using the button with the beetle icon.



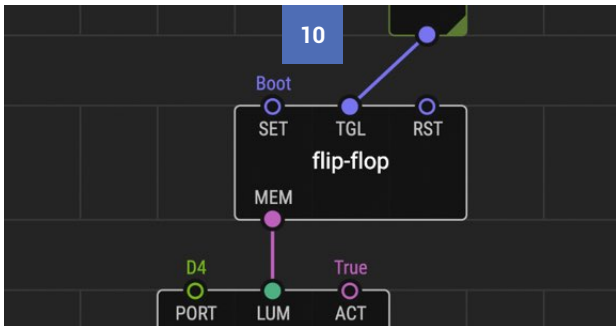
## TWEAK AND WATCH

Press the **tweak-pulse** node and watch what happens to the LED. Each time you press, the light should flash 5 times.



## FLIP-FLOP NODE

Now lets try a different flip node. Delete the **flip-n-times** node and add a **flip-flop** node (*xod/core*).



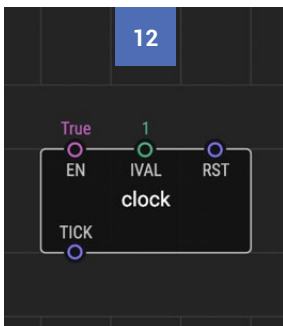
### CONNECT AND SET FLIP-FLOP PINS

The **flip-flop** node has 3 input pins: SET, RST and TGL (toggle). TGL switches the node between true and false each time it receives a pulse. Connect the **tweak-pulse** node to TGL. Set SET to 'On Boot' and RST to 'Never'. The MEM (memory) output pin reads out the latest state of the node. Connect this to the LUM pin of **led**.



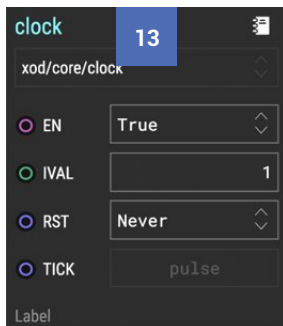
### TEST THE PATCH

Upload the program and pulse the **tweak-pulse** node. The LED should switch between on and off each time you press it.



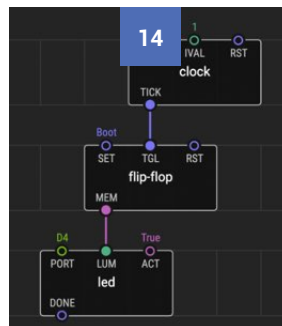
### CLOCK NODE

The **flip-flop** node can also flash the LED when combined with a **clock** node. Add a **clock** node (**xod/core**) to the patch.



### SET CLOCK PINS

Set EN (enabled) to 'True' and RST to 'Never'. IVAL determines how often the clock ticks (in secs). Set this to '1'.

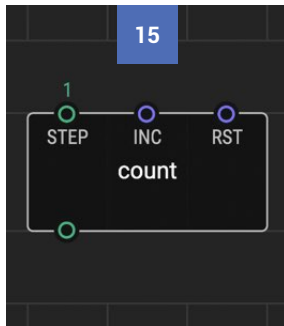


### RECONNECT

Delete the **tweak-pulse** node and connect the **clock** node in its place by linking the TICK pin to the TGL pin.

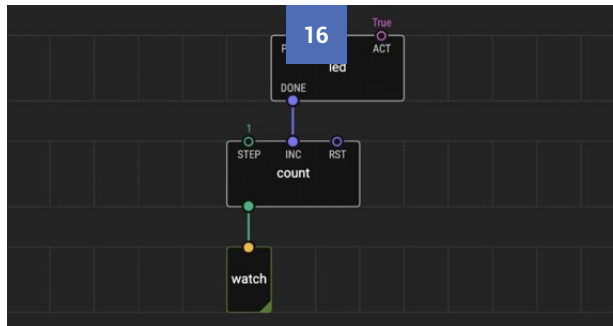


# Flip, Clock and Count nodes



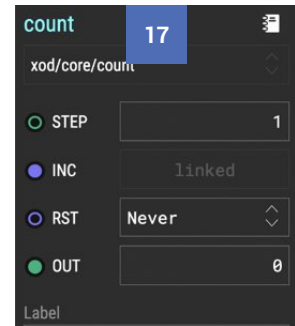
## COUNT NODE

We'll also add a **count** node to this patch so that we can monitor the number of times the LED flashes. Add a **count** node (`xod/core/count`).



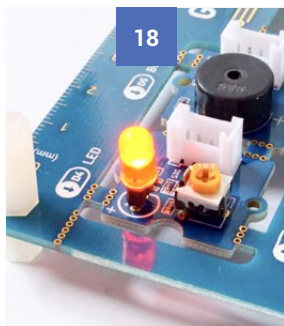
## CONNECT COUNT NODE

Connect the **led** DONE pin to the **count** INC (increase) pin. The DONE pin pulses each time the LED turns on or off, and the INC pin increases the count each time it is pulsed. Connect the **count** output pin to a **watch** node so we can see the count. This will let us see on the screen each time the LED pulses.



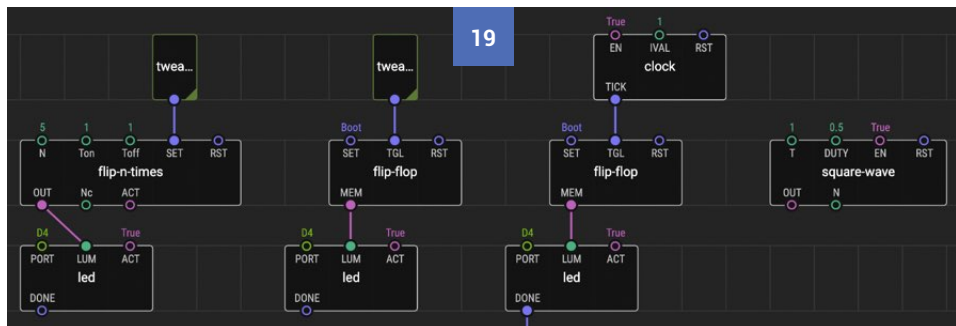
## SET COUNT PINS

Set RST to 'Never'. STEP determines how much the count increases by with each pulse. Set this to '1'.



## TEST THE PATCH

Upload the program. Watch the LED and count. The LED should flash and the count will increase with each flash.



## EXPERIMENT!

As with all programming, there is always more than one way to achieve a similar outcome, and different methods may suit different applications. Here we have tested two different ways of making the LED flash, but there are plenty of other ways you can experiment with.

Why not try using a **square-wave** node to make the LED flash? See if you can work it out using the help pane and XOD website. Or you can just try playing around with the nodes you've already tried. Try experimenting with different timings and patterns of flashing.

## Watching Pulse Pins: Combining Count and Watch Nodes

Combining **count** and **watch** nodes is a really useful way to visualise the output of a pulse pin. Unfortunately, a pulse output can not be directly connected to a **watch** node in XOD, as the data types (pulse and string) clash.

We can get around this by connecting a pulse output to the INC pin of a **count** node, and then connecting a **watch** node to the **count** output pin (as we did in **Steps 15-16** of this Task). In this setup, the **count** node increases with each pulse sent, and we can visualise this in the debugger with the **watch** node.



## Discovering New Nodes

The last step of this task encourages you to try out a new node: the **square-wave** node. We have not provided specific information about this node here, but it is useful to practice discovering new nodes for yourself.

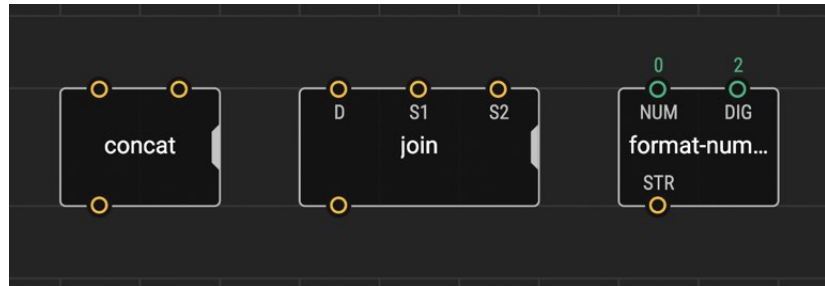
To work out how a new node works, it is best to start with understanding what its pins do. In most cases, the Quick Help pane will give a brief explanation of the node and what each of its pins does. This is often enough information to work out the node's function and how to use it.

You can also find additional documentation about each node on the XOD website. You can access this by clicking on the document button next to the node name in the Inspector pane. Or by visiting [www.xod.io/libs](http://www.xod.io/libs) and using the search function.

When the documentation for a node is not sufficient, you can usually still work out its function by adding **tweak** nodes to its inputs and **watch** nodes to its outputs. Then simulate or 'Upload and Debug' your patch. Try editing each of the inputs in turn and watch how this affects the outputs. In this way you can often determine a node's function experimentally.

If you are still having trouble, you can always find help on the XOD forum at [www.forum.xod.io](http://www.forum.xod.io).

# Concat, Join and Format-Number nodes



## CONCAT NODE

The **concat** node allows you to join two or more sets of strings together. This is useful for combining different inputs for display or storage. E.g. combining a number readings from a sensor with the symbol for it's units. **Concat** will join the inputs directly, so if you require a space between them you will need to input this in your string.

Concat is a variadic node. Meaning you can change the number of inputs. You can do this by dragging out the white tab on the right side of the node.

## JOIN NODE

The **join** node is similar to the **concat** node, but has the additional feature of allowing you to chose how the different string inputs are joined together. The delimiter (D) pin determines what character is used to separate inputs, e.g. a space, comma or colon etc. This is particularly useful for storing data readings, as you can separate values with a comma or tab to create comma-separated (.csv) or tab-separated (.tsv) files. Like **concat**, the **join** node is variadic, and can take as many inputs as necessary.

## FORMAT-NUMBER NODE

As the name suggests, the **format-number** node allows you to format number outputs. With this node, you can determine the amount of decimal places displayed, which can be very useful if you would like to display a sensor reading, for example. This node also converts the format of the input from a number to a string.

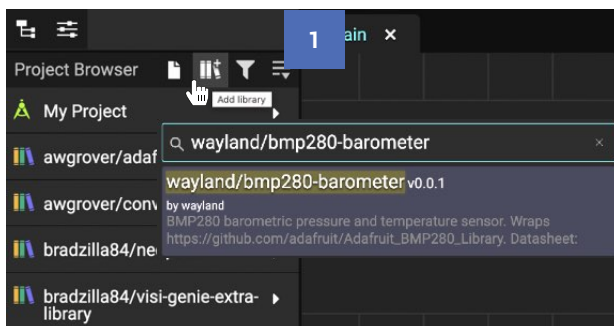
Other nodes useful for formatting numbers include **number-split-to-digit**, from the library **gst/number-split-to-digit**), **dec-to-2digits** and **dec-to-4digits**, both from the library **cesars/utills**.

## Task 5: Requirements

- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (air pressure sensor module)
- USB-A to micro USB cable

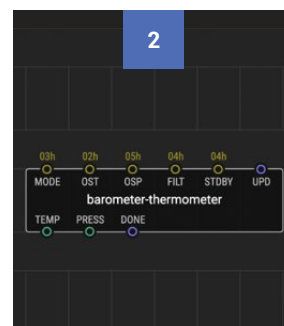
In this task we'll look at using **concat**, **join** and **format-number** nodes. These nodes are especially useful when working with data and displays.

The **concat** and **join** nodes are both used to combine information in the form of strings (text). The **format-number** node is used to set the number of decimal points displayed in a number. In combination, these nodes are useful for formatting the outputs of sensor modules, both for data storage, and for display on a screen.



### NEW PATCH AND ADD LIBRARY

Open a new patch (or move on to tuto501). To work with the air pressure sensor (also known as a barometer) you will need to install the library **wayland/bmp280-barometer**.

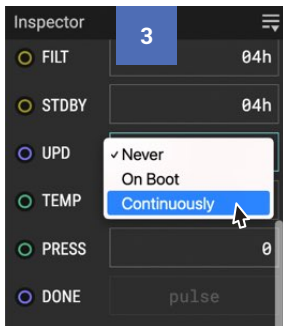


### ADD BAROMETER-THERMOMETER NODE

Add a **barometer-thermometer** node (**wayland/bmp280-barometer**) to the patch.

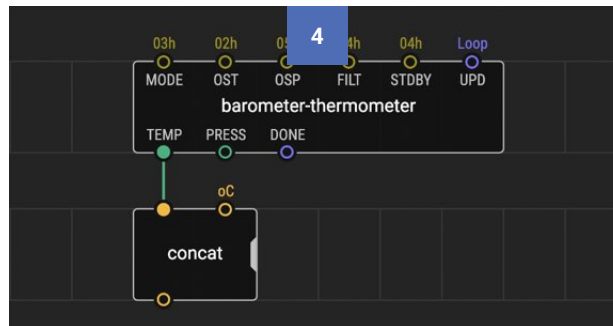


# Concat, Join and Format-Number nodes



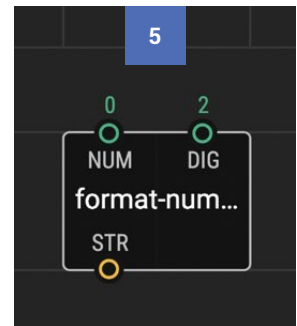
## SET BAROMETER-THERMOMETER NODE

Set UPD to 'Continuously'. Leave other inputs as they are. Outputs are temperature (TEMP) and pressure (PRESS).



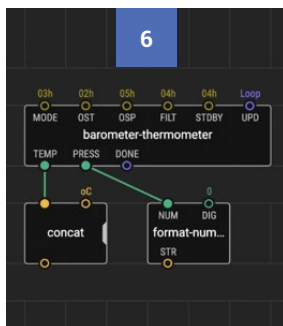
## ADD AND SET CONCAT NODE

Add a **concat** node (**xod/core**). This node combines multiple strings from the input pins into a single output. This node is 'variadic' meaning you can expand the node by pulling on the tab on the right, letting you increase the number of inputs. Connect the first input to the TEMP pin. Set the second input to 'oC' (degrees centigrade).



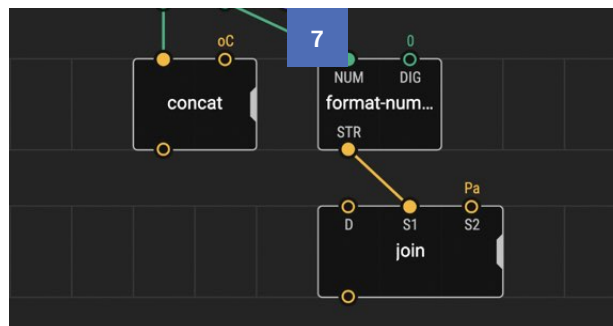
## ADD FORMAT-NUMBER NODE

Add a **format-number** node (**xod/core**). This node lets you format the number of decimal places in a number.



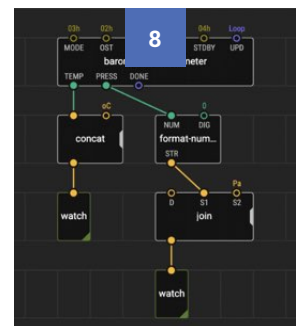
## SET FORMAT-NUMBER NODE

Link the **barometer-thermometer** PRESS pin to the **format-number** NUM (number) pin. Set DIG (digits) to '0' decimal places.



## ADD AND SET JOIN NODE

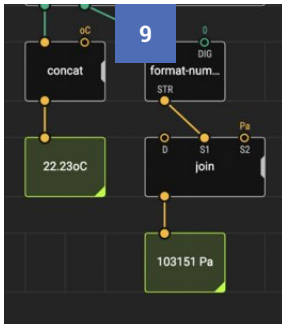
Add a **join** node (**xod/core**). This is similar to **concat**, but has a D (delimiter) pin. D determines how inputs are joined (e.g. via a space or colon). It's automatically set to be a space. Leave it as this. Connect the first input (S1) to the **format-number** STR (string) pin. Set the second input to 'Pa' (pascals, the unit of air pressure).



## ADD WATCH NODES

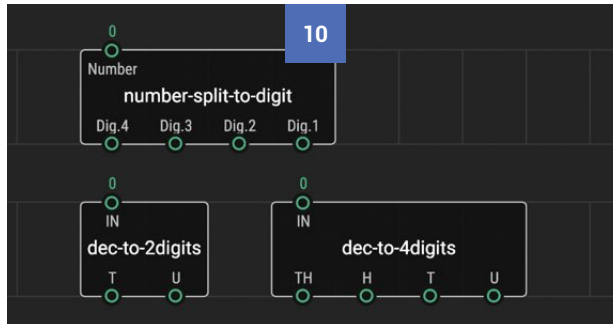
Add two **watch** nodes (**xod/debug**). Link one to the output of the **concat** node, and one to the output of the **join** node.





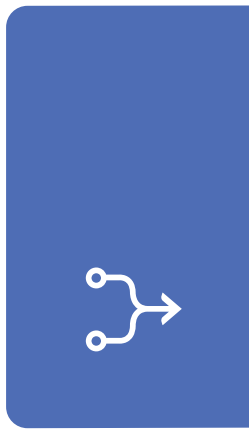
### TEST THE PATCH

Upload and debug. Look at the **watch** nodes. You may need to expand the **watch** nodes by pulling on the bottom right corner.



### EXPERIMENT!

Play around with the nodes in your patch to see how you can format the sensor output in different ways. Try exploring other nodes available for formatting numbers in XOD. For example, **number-split-to-digit** (from **gst/number-split-to-digit**), **dec-to-2digits** or **dec-to-4digits** (both from **cesars/utils**).



## More Information on Basic Nodes

This section has covered a number of basic nodes that we have found useful in building simple biological devices. However, there are plenty of other useful nodes out there, both pre-installed in XOD, and created by XOD users such as yourself.

In the rest of this guide we will continue to explore useful nodes and techniques in XOD, but if you'd like to explore for yourself, here are a few useful resources for getting to grips with XOD:

### XOD TUTORIAL

Each time you open XOD it will offer you the option of following its inbuilt tutorial. Working through this is a great way to learn more about what XOD can do.

It is also available online at [www.xod.io/docs/tutorial](http://www.xod.io/docs/tutorial).

### XOD GUIDE

The XOD user guide provides advice on some more complex concepts, as well as some case studies to work through. It is available online at [www.xod.io/docs/guide](http://www.xod.io/docs/guide).

### XOD CORE LIBRARY

Taking a look through the nodes in the XOD core library will help you understand the most basic nodes available and what they do. Find **xod/core** in the Project Browser.



# Lesson 4: Building Devices



**Creating new nodes**



**Using Buses**



**Program logic**



**Sequences and Loops**





# Creating new nodes

## Why Create New Nodes?

If we want to build programmes capable of more complex functions than producing a simple input and output, we will need to add a few more skills to our repertoire. More complex programmes often require more nodes, and the multitude of nodes and links can quickly become confusing. The good news is that XOD provides several ways of reducing the complexity of your patches and keeping your programmes neat and tidy. This is good practice so that you can keep track of what you're doing, and also for others who may need to understand your programme.

One way to simplify a complex programme is to make your own nodes. This means that you can encapsulate specific functions within your programme into neat little packages that can be easily connected to each other. It also has the advantage that they can easily be shared with the wider XOD community, making useful new nodes available to everyone.



XOD terminal nodes

Creating your own nodes is much easier than it sounds. It is essentially the same as creating any other patch, but we need to add special nodes called 'terminals' to allow our new node to communicate with other nodes.

There are two types of terminals: inputs and outputs. Like *tweak* nodes, they come in different types based on their data type. The nodes above from left-to-right, top-to-bottom are: **input-boolean**, **input-byte**, **input-number**, **input-port**, **input-pulse**, **input-string**, **input-t1** (custom input type), **output-boolean**, **output-byte**, **output-number**, **output-port**, **output-pulse**, **output-string**, **output-t1** (custom input type).

In the next task we'll explore in more detail how to use these terminals to create your own nodes, using the inbuilt OLED screen on your board as an example.

The XOD website also provides some excellent information about how to make your own nodes at [www.xod.io/docs/guide/nodes-for-xod-in-xod](http://www.xod.io/docs/guide/nodes-for-xod-in-xod).



# Creating new nodes

## Task 6: Requirements

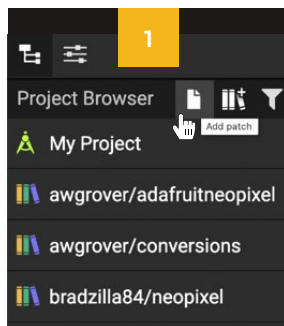
- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (sound sensor and OLED screen modules)
- USB-A to micro USB cable

In this task we'll learn how to make a new node that will allow us to write text on our OLED screen.

Instructing the OLED screen to display text is a slightly more complex task than we have done so far, and involves several nodes to represent the screen rather than one.

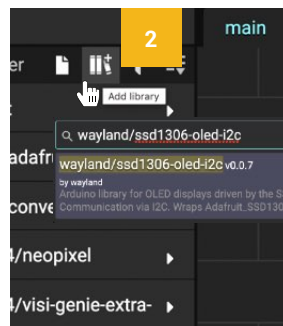
In this task we'll be combining these multiple nodes into one, which we will then use to display the readings from our onboard sound sensor.

Creating nodes like this is useful as it helps to simplify the patch, and we can also save new nodes for later use in different programmes.



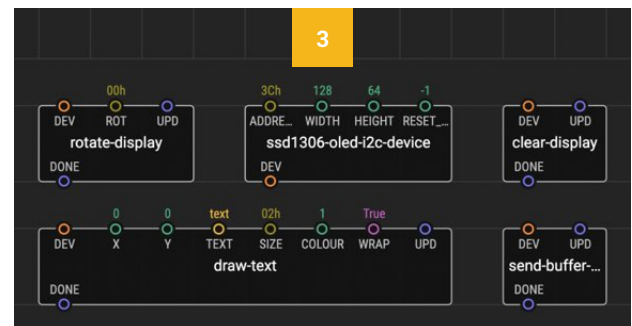
### NEW PROJECT AND NEW PATCH

Save your project and create a new one (or move on to tuto601). Add a new patch to the project, and name it **'write-text-to-oled'**.



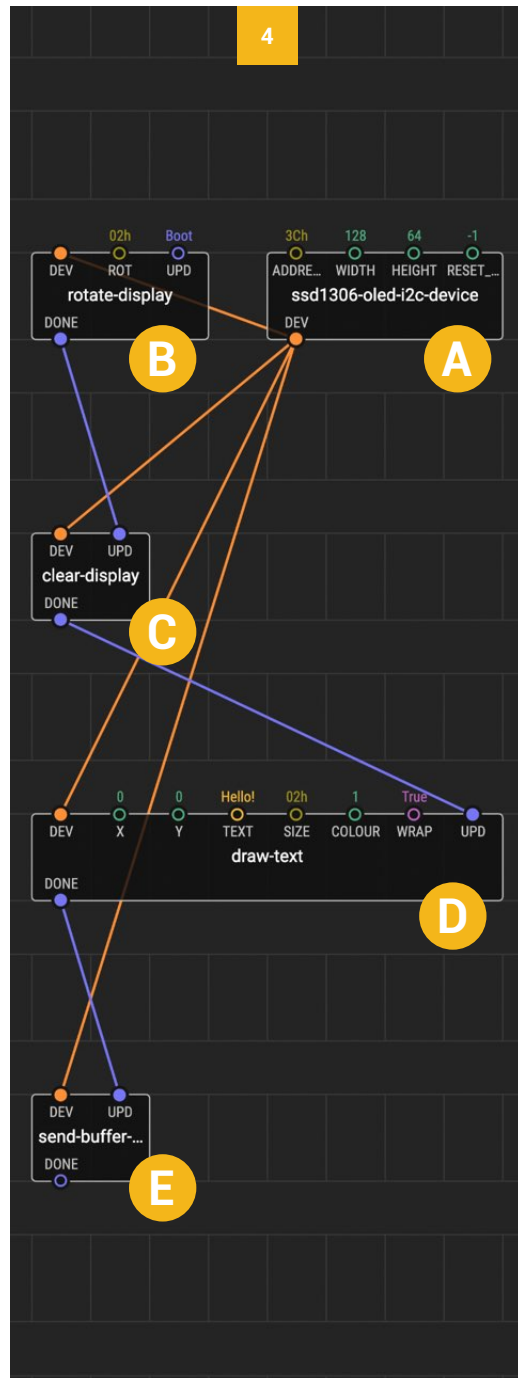
### ADD LIBRARY

To work with the OLED screen you will need to install the library **wayland/ssd1306-oled-i2c**.



### ADD NODES

From the **wayland/ssd1306-oled-i2c** library, add the following nodes to your patch: **ssd1306-oled-i2c-device**, **rotate-display**, **clear-display**, **draw-text**, **send-buffer-to-display**.



## SETTING UP THE OLED SCREEN

Setting up the OLED screen requires several connections between these nodes, so let's take it step by step.

### A

#### SSD1306-OLED-I2C-DEVICE

This node represents the OLED device. WIDTH and HEIGHT set the dimensions of the screen in pixels. Leave these as '128' and '64'. ADDRESS identifies the port, leave this as '3Ch'. RESET represents the screen's reset pin. Leave this as '-1' as our board does not have a dedicated screen reset pin. **The output of this node, DEV (device) needs to be connected to each of the other nodes' DEV input pins.**

### B

#### ROTATE-DISPLAY

You can change the screen orientation using this node. Set ROT to '02h', which is correct for our screen. Set UPD to 'On Boot' so that the screen updates when the programme starts. This node starts a sequence that allows us to display items on the screen. **In this sequence, each DONE pin connects to the next UPD pin.** Connect the **rotate-screen** DONE pin to the **clear-display** UPD pin.

### C

#### CLEAR-DISPLAY

This node should be used before displaying anything on the screen. Connect the DONE pin to the **draw-text** UPD pin.

### D

#### DRAW-TEXT

This node inputs the text we want displayed. X and Y determine the position of the text by coordinates. Leave these as '0', '0'. TEXT is where you enter your text. Use 'Hello!' as a test. SIZE determines the size of the text. Leave this as '02h'. COLOUR determines the colour of the text (black or white). Set this to '1'. WRAP determines whether the text is wrapped within the boundaries of the screen. Leave this as 'True'. Connect the DONE pin to the **send-buffer-to-display** UPD pin.

### E

#### SEND-BUFFER-TO-DISPLAY

So far we have written information to the microcontroller's memory, but we haven't actually sent it to the screen. This node is the final step that sends this data. It needs to be used whenever you want to display something on the screen.

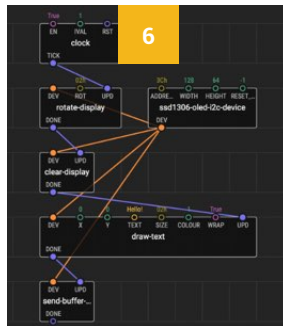


# Creating new nodes



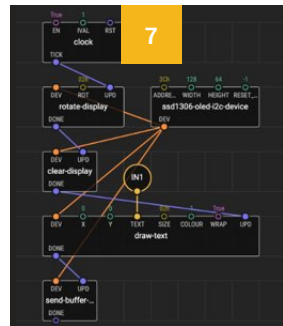
## TEST THE PATCH

Upload the patch and watch your OLED screen. White text should appear in the top left-hand corner of the screen.



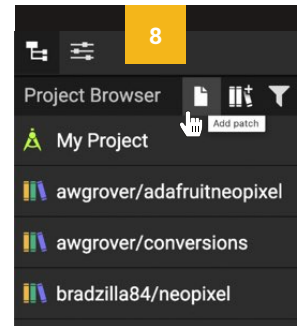
## ADD A CLOCK NODE

Add a **clock** node (**xod/core**) and link the TICK pin to **rotate-display** UPD. This will make your screen update once a second.



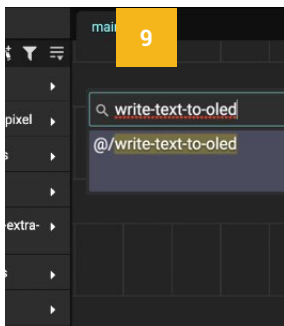
## ADD AN INPUT-STRING NODE

Add an **input-string** node (**xod/patch-nodes**) and connect it to **draw-text** TEXT.



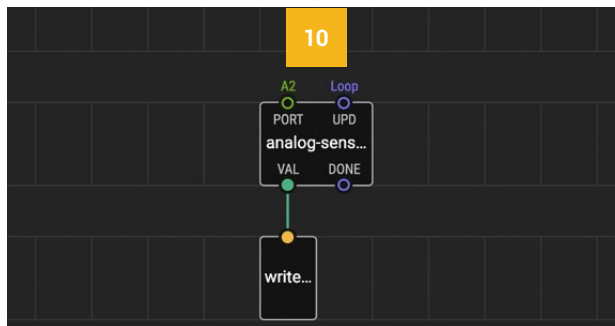
## MAKE A NEW PATCH

Now we've made our new node, let's try adding it to another patch. Add a new patch and name it 'sound-sensor'.



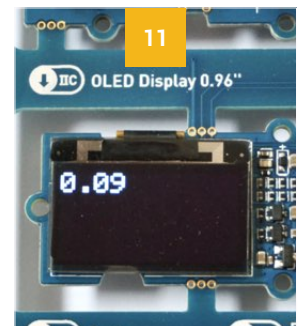
## ADD WRITE-TEXT-TO-OLED NODE

To add your new node to a patch, simply search for it as usual, or drag the patch from the Project Browser into the patch.



## ADD AND SET ANALOG-SENSOR NODE

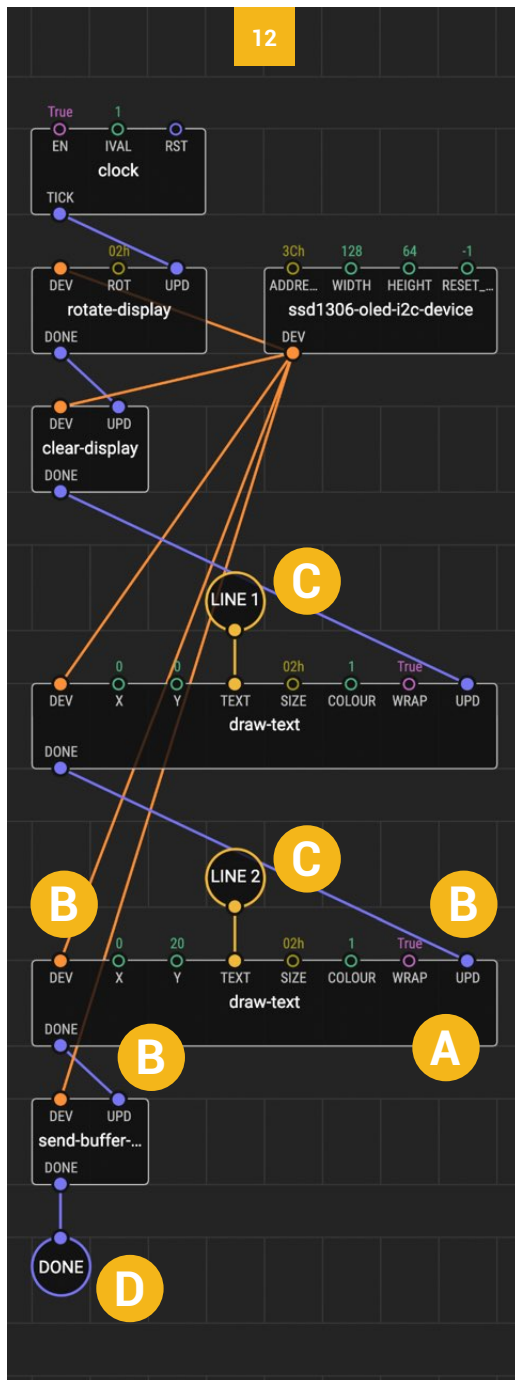
Now that we have a node that will write text to the screen, let's use it to display a sensor reading. For many common analog sensors (including the inbuilt sound sensor) you can use the simple XOD **analog-sensor** node (**xod/common-hardware**). Add this node, set PORT to 'A2', and connect VAL to the **write-text-to-oled** node.



## TEST THE PATCH

Upload the patch, and you should see the readings from the sound sensor displayed on your screen.





## MODIFY YOUR NODE

Writing a line of text directly to the OLED screen is great, but what if we need a more complicated node? For example, one that takes multiple lines of text, or one that sends a signal once the text has been uploaded?

Lets go back to our **write-text-to-display** node. You can do this by opening the patch in the Project Browser, or by double clicking on the node in your patch.

Make the changes listed below to expand the capabilities of your node.

### A

#### SECOND DRAW-TEXT NODE

Add a second **draw-text** node below the first. We will use this to draw a second line of text. Set the Y pin of this node to 20. This will move the text down by 20 pixels, creating a new line.

### B

#### RECONNECT

Delete the link between the first **draw-text** node and **send-buffer-to-display**. Link the first **draw-text** DONE pin to the second **draw-text** UPD pin. Link the second **draw-text** DONE pin to the **send-buffer-to-display** UPD pin. Connect **ssd1306-oled-i2c-device** DEV to the second **draw-text** DEV pin.

### C

#### LABEL INPUT-STRING NODES

Add a second **input-string** node and connect it to the TEXT pin of the second **draw-text** node. We will now have more than one input into our node, so we need give them labels, to avoid confusion. Click on each **input-string** node in turn. Use the Inspector pane to name your nodes by typing in the 'Label' text box. Name your nodes 'LINE 1' and 'LINE 2' respectively.

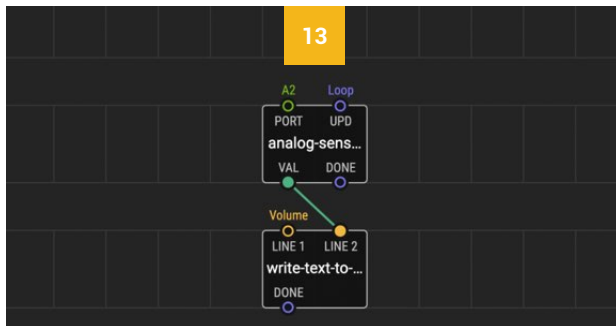
### D

#### ADD OUTPUT-PULSE NODE

Add an **output-pulse** node to the patch, and connect it to the **send-buffer-to-display** DONE pin. Use the Inspector to label this node 'DONE'.

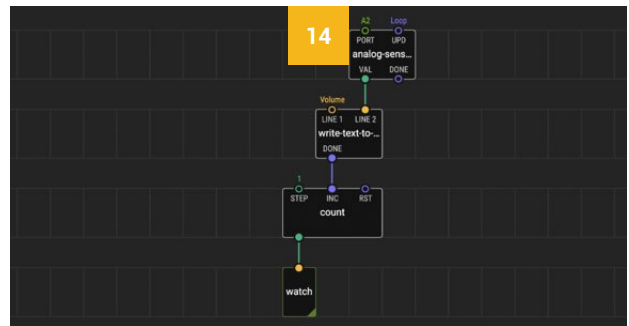


# Creating new nodes



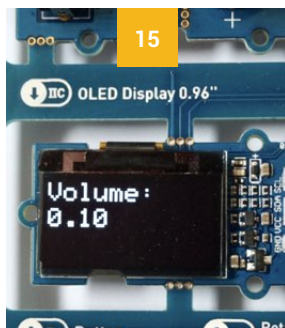
## MODIFYING YOUR PATCH

Return to your 'sound-sensor' patch using the tab at the top, or the Project Browser. You will notice that the **write-text-to-oled** node has changed. It now has two inputs and an output. Delete the link between VAL and LINE 1 and link VAL to LINE 2 instead. Use the Inspector to set LINE 1 to 'Volume:'.



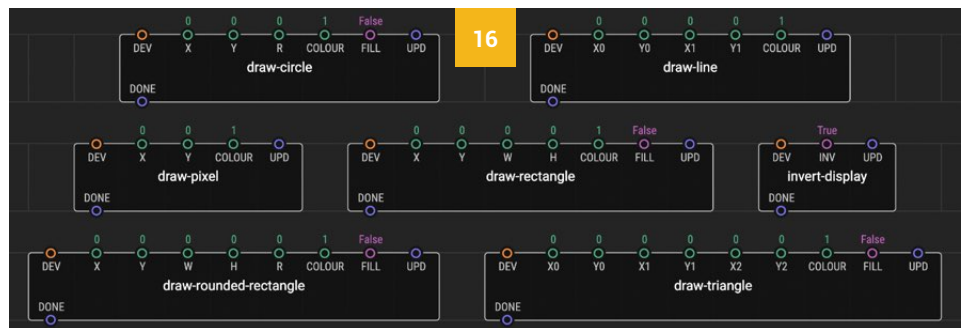
## ADD COUNT AND WATCH NODES

Add a **count** node (**xod/core**) and **watch** node (**xod/debug**). Link the **write-text-to-oled** DONE pin to the **count** INC pin, and the **watch** node to the **count** output pin. A **watch** node cannot be directly linked to a pulse pin, so this is a useful trick if you want the watch the output from a pulse pin.



## TEST THE PATCH

Upload and debug. You should see two lines of text on your screen, and the watch node will count when the screen updates.



## EXPERIMENT!

The OLED screen is a really useful device, and can be used in a multitude of different ways. Try playing around with your new node by adding another line, or changing the position and size of the text. Or you can try using the OLED to display data from a different sensor. You can also experiment with some of the other nodes in the **wayland/ssd1306-oled-i2c** library. This library contains lots of useful nodes for drawing different objects on the screen, as well as several example patches to show you how they work.

## Sharing Nodes and Publishing Libraries

One of the great advantages of XOD is the growing community of contributors, who are generating an ever-expanding range of nodes and libraries for other to use. When you create new nodes that you think might be useful for others, you can easily share these with the XOD community by publishing them as a library.

There are no strict rules about what constitutes a library, so even if you only create one node, this can still be a library. Publishing allows others to use your nodes, but is also useful for reusing your own work, as you can download your own libraries for use in all of your projects.

Creating a library is essentially just making a project with a patch for each node. You can also include patches with example of how to use the nodes, as there are in the **wayland-ssd1306-oled-i2c** library. Once you've created your project, you need to set the metadata. Do this by navigating to 'Edit > Project Preferences' in the menu bar. Here you should enter a name a description for you library, as well as a licence type (e.g. GNU, CC-BY etc. more info. at [www.opensource.org/licenses](http://www.opensource.org/licenses)).

To publish your library, go to 'File > Publish Library' in the menu, click 'Publish', and you're done! You can find out more about publishing libraries at [www.xod.io/docs/guide/creating-libraries](http://www.xod.io/docs/guide/creating-libraries).

## Documenting Nodes

When publishing your work it is good practice to make sure that your nodes are well documented. This helps to remind yourself what you've done, and allows others to get an idea of how the node can be used.

Before you publish you library, make sure that you have described the node and each of it's pins. To write a description of the node, click a blank space on the patch and a 'Description' box will appear in the Inspector pane. Write a brief description about what the node does and what its used for, e.g. *"This node writes two lines of text to the ssd1306 OLED screen"*.

To write a description of the pins, click on an *input* or *output* node and you will see the 'Description' box at the bottom of the Inspector pane. Write a brief description of the pin, e.g. *"String to display on the first line of text. Text will appear at coordinates 0:0"*.

You can also provide more information about how your node works by adding comments to the patch. To do this, navigate to 'Edit > Insert Comment' in the menu bar. You can find out more about documenting nodes at [www.xod.io/docs/guide/documenting-nodes](http://www.xod.io/docs/guide/documenting-nodes).



# Using Buses

## Task 7: Requirements

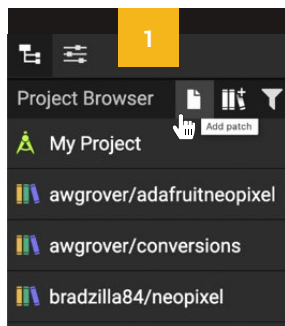
- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (3-axis acceleration and OLED screen modules)
- USB-A to micro USB cable

In this task we'll look at another way to simplify our patches: using buses. Buses are a way to link pins 'invisibly' so that you don't have too many link intersections that make the data flow confusing.

Buses are a little like **input** and **output** nodes. They come in two types, **to-bus** and **from-bus**, and they automatically take the data type of the pin they're connected to.

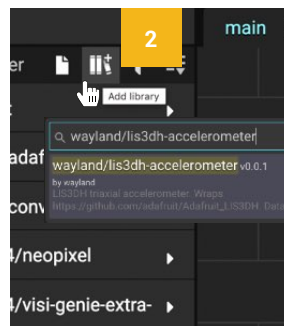
The **to-bus** node is used like an **output** node and sends information from an output to a bus. The **from-bus** node acts like an **input** node and retrieves information from the bus of the same name.

We'll practice using buses by displaying the output of our 3-axis acceleration sensor (also known as an accelerometer or tilt sensor) on our OLED screen.



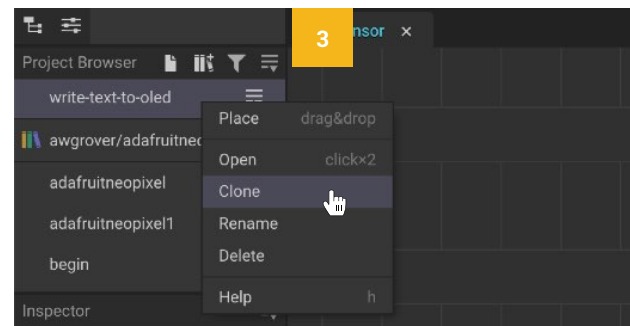
### MAKE A NEW PATCH

Add a new patch to the project, and name it **'tilt-sensor'** (or move on to tuto701).



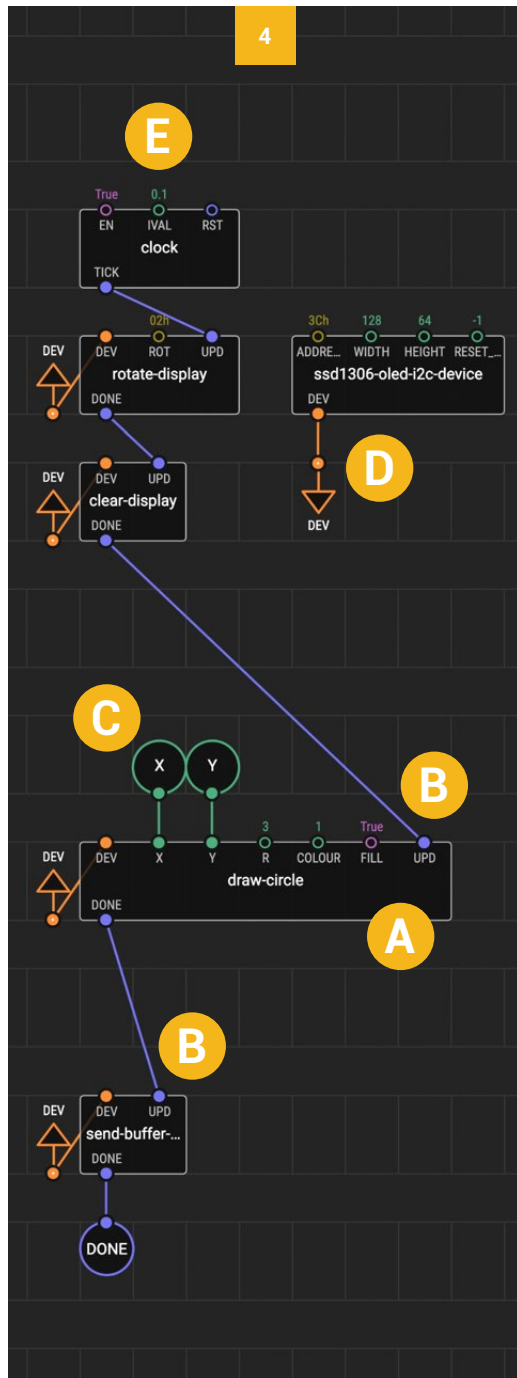
### ADD LIBRARY

To work with the accelerometer you will need to install the library **wayland/lis3dh-accelerometer**.



### CLONE WRITE-TEXT-TO-OLED PATCH

For this task we'll be using the OLED display again, but in a slightly different way. So that we don't have to start again, we can clone the **write-text-to-oled** patch by right clicking on the patch in the Project Browser and selecting 'Clone'. Rename the new patch from **'write-text-to-oled-copy'** to **'write-dot-to-oled'**.



## MODIFY YOUR NODE

This time we would like to draw a small circle on the screen instead of text. The circle will move around the screen as you tilt the board.

Follow the instructions below to modify the **write-dot-to-oled** patch for this new purpose.

### A

#### ADD AND SET DRAW-CIRCLE NODE

Delete both **draw-text** nodes along with their associated **input** nodes. Add a **draw-circle** node. Set R (radius) to '3' to make a circle 3 pixels wide. Leave colour as '1'. Set FILL to 'True' so that we get a solid circle rather than an outline.

### B

#### RECONNECT

Link the **clear-display** DONE pin to the **draw-circle** UPD pin. Link the **draw-circle** DONE pin to the **send-buffer-to-display** UPD pin.

### C

#### ADD INPUT-NUMBER NODES

Add two **input-number** nodes and connect them to the X and Y pins. Use the inspector to label these 'X' and 'Y'.

### D

#### REPLACE LINKS WITH BUSES

Although the OLED patch worked, it was very messy, with links criss-crossing, and it would be easy to miss a connection. Let's improve this by replacing these links with a bus. First, delete all of the orange links between the **ssd1306-oled-i2c-device** and the other nodes. Add a **to-bus** node (**xod/patch-nodes**) and link it to the output of **ssd1306-oled-i2c-device**. Use the inspector to label this node 'DEV'. Add a **from-bus** node (**xod/patch-nodes**). Make sure this **from-bus** node is also labelled 'DEV' as buses can only communicate if they have the same name. Repeat this process, or copy and paste the DEV **from-bus** node until you have four in total. Connect these to each of the DEV input pins on the nodes **rotate-display**, **clear-display**, **draw-circle** and **send-buffer-to-display**.

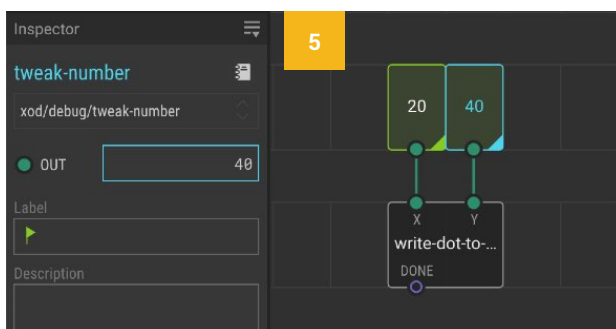
### E

#### CHANGE CLOCK TIMING

Set the **clock** IVAL pin to 0.1, so that it updates more frequently.

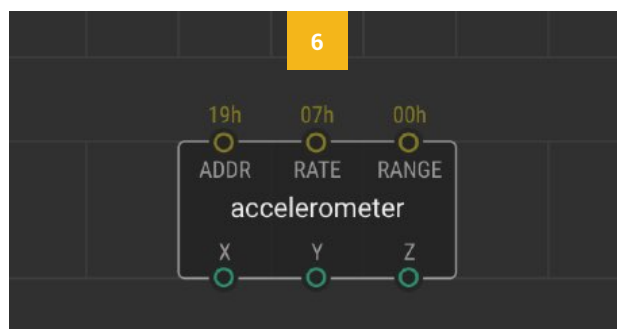


# Using Buses



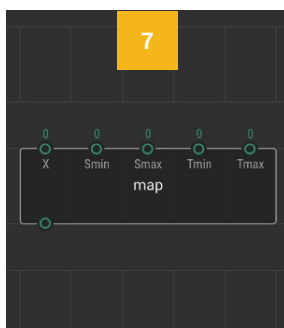
## TEST THE NODE

Test the node by returning to your **tilt-sensor** patch, adding a **'write-dot-to-oled'** node, and connecting two **tweak-number** nodes to the X and Y inputs. Upload and debug. Click on the *tweak-number* nodes and use the Inspector to change their values. Watch how this shifts the dot around the screen.



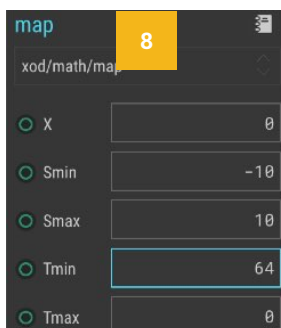
## ADD AN ACCELEROMETER NODE

Add an **accelerometer** node (**wayland/lis3dh-accelerometer**). We want to use the output from the accelerometer to set the location of the dot on the screen. You could connect the **accelerometer** X and Y pins directly to the **write-dot-to-oled** X and Y pins, but it wouldn't work as the nodes' ranges don't match up.



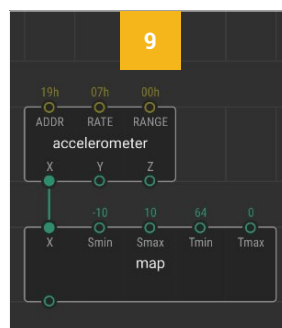
## ADD A MAP NODE

To fix this, add a **map** node (**xod/math**). This node lets us map the **accelerometer** output range to the **write-dot-to-oled** input range.



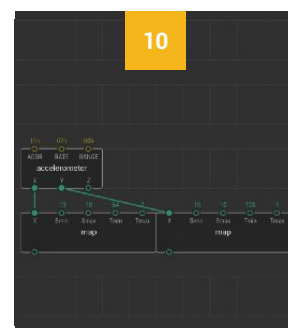
## SET MAP NODE

Set Smin to '-10' and Smax to '10' (the range of the **accelerometer**). Set Tmin to '64' (the screen height) and Tmax to '0'.



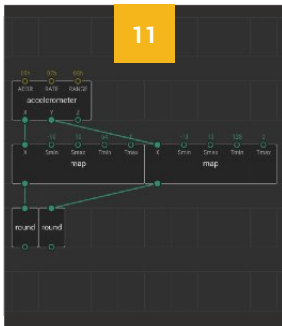
## CONNECT MAP NODE

Link the **accelerometer** X and **map** X pins. The node now converts the **accelerometer** X range to values within the height of the screen.



## REPEAT MAP NODE

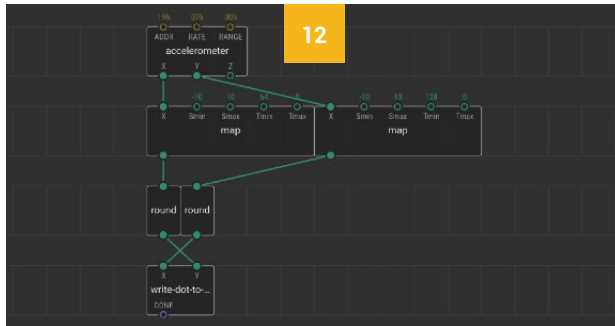
Add a second **map** node add link it to the Y output. Set Smin to '-10', Smax to '10', Tmin to '128' (the screen width) and Tmax to '0'.



11

### ADD ROUND NODES

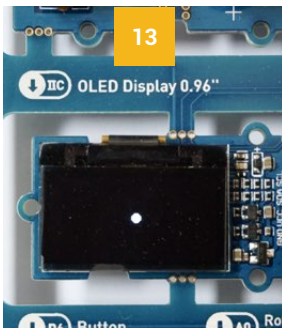
Add two **round** nodes (**xod/math**). Connect them to the outputs of the **map** nodes. This will round the outputs to whole numbers.



12

### CONNECT ACCELEROMETER TO OLED

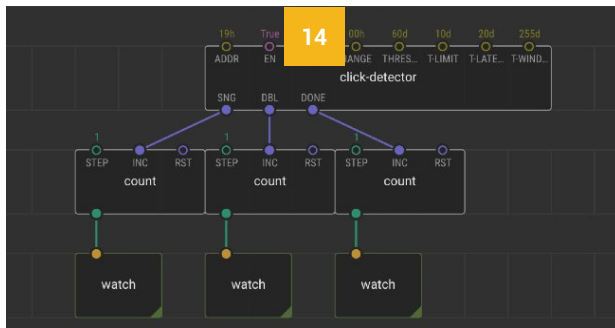
Delete the **tweak** nodes from the **write-dot-to-oled** node. Connect the **round** output linked to the **accelerometer** X output to the Y input pin, and the **round** output linked to the **accelerometer** Y output to the X pin. This seems counter-intuitive, but is due to the settings of the different nodes.



13

### TEST THE PATCH

Upload the patch. Tilt your board left and right, backwards and forwards, and watch the little dot on the screen move!



14

### EXPERIMENT!

Experiment with the nodes in this patch. Can you get some text to appear when the dot lands in the middle? It is also worth exploring the **wayland/lis3dh-accelerometer** library, as it has useful nodes and plenty of demonstrations. For example, the **click-detector** node above, which detects taps of the sensor.





# Program logic

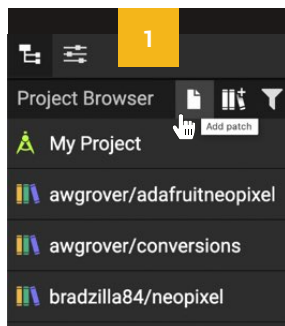
## Task 8: Requirements

- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (light sensor and OLED screen modules)
- USB-A to micro USB cable

In this task we'll take some of the skills we've learned so far in this lesson and use them to create a more complex programme that uses logic to instruct the board what to do.

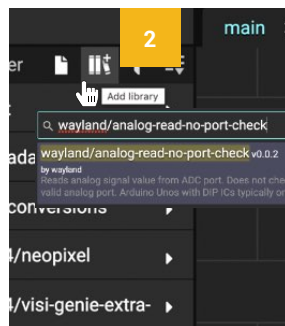
We'll be using the light sensor and OLED screen modules of the board to create a simple light sensing device.

Whilst this is still a fairly simple device, it contains a lot of the basic functions that you can use for your own instruments: an input in the form of a sensor; a logic programme that instructs the board what to do based on the value of this input; and an output that changes something, in this case the text displayed on a screen.



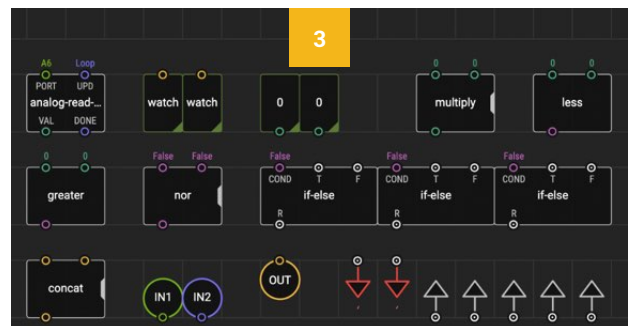
### MAKE A NEW PATCH

Add a new patch to the project, and name it '**light-sensor**' (or move on to tuto801).



### ADD LIBRARY

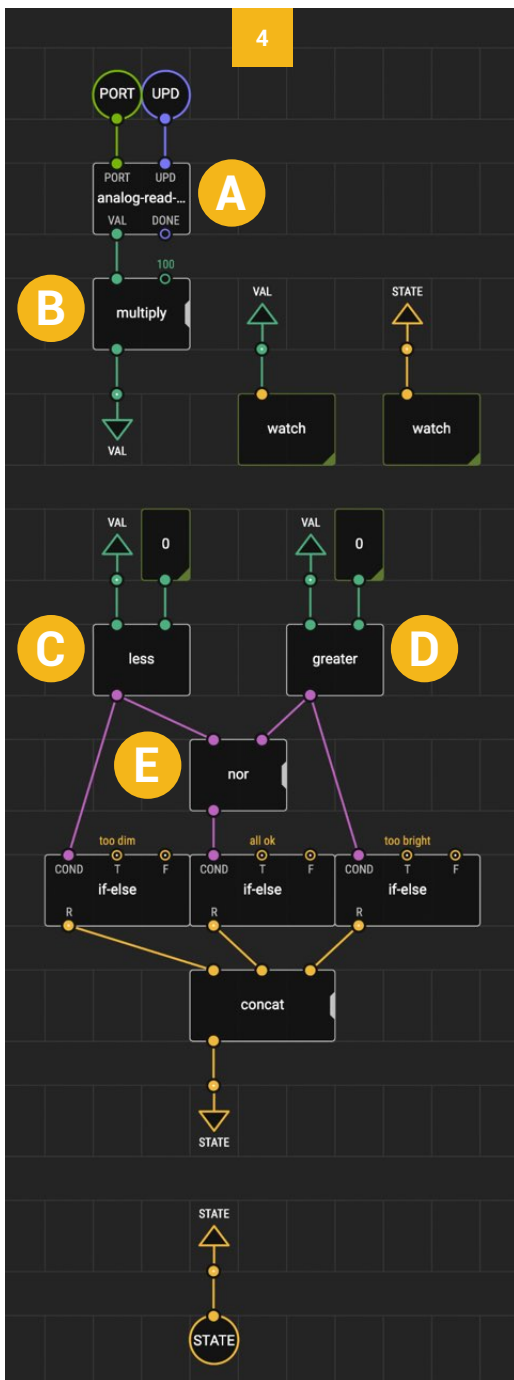
To work with the light sensor you will need to install the library **wayland/analog-read-no-port-check**.



### ADD NODES

Add the following nodes to the **light-sensor** patch: **analog-read-no-port-check** (**wayland/analog-read-no-port-check**), **watch** x2, **tweak-number** x2 (**xod/debug**), **multiply**, **less**, **greater**, **nor**, **if-else** x3, **concat** (**xod/core**), **input-port**, **input-pulse**, **output-string**, **to-bus** x2, **from-bus** x5 (**xod/patch-nodes**).





## SET UP YOUR LIGHT-SENSOR NODE (PART 1)

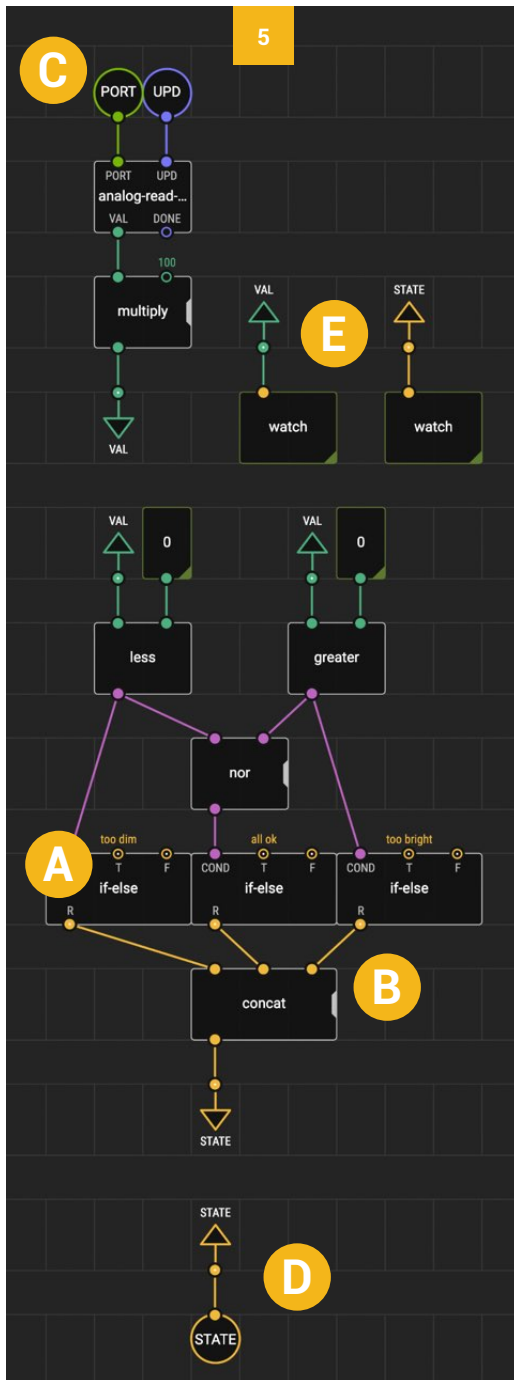
We want our node to return one of three readings depending on the light intensity: 'too dim', 'all ok' or 'too bright'. We'll use a combination of simple logic functions to programme this capability.

Follow the instructions below to set up your patch.

- A ANALOG-READ-NO-PORT-CHECK**  
This node represents the light sensor. Connect the VAL output to the first input of the **multiply** node.
- B MULTIPLY**  
We will use the **multiply** node to scale up the output of the *sensor* node. The node will multiply the input values, so let's set the second input '100'. Connect one of the *to-bus* nodes to the **multiply** output and label it 'VAL'.
- C LESS**  
This node takes the first input value and compares it to the second input value. It will return 'True' if the first value is less than the second, and 'False' if not. Connect one of the *from-bus* nodes to the first input and label it 'VAL' so it receives the value output by the **multiply** node. Connect one of the *tweak-number* nodes to the second input so you can tweak the lower limit later.
- D GREATER**  
The **greater** node is very similar to the **less** node, but it will return 'True' only when the first input is greater than the second input. Repeat the same connections for the **greater** node. Connect a *from-bus* node to the first input and label it 'VAL'. Then connect a *tweak-number* node to the second input to let you tweak the upper limit.
- E NOR**  
The **nor** node will only return 'True' if both inputs read 'False'. We want a third state that triggers if the light intensity is neither less than the lower limit, nor greater than the upper limit, and we will use the **nor** node to achieve this. Connect the output of **less** node to the first **nor** input, and the output of the **greater** node to the second **nor** input.



# Program logic



## SET UP YOUR LIGHT-SENSOR NODE (PART 2)

You should now have completed the first half of the patch. Continue setting up the second half by following the instructions below.

**A**

### IF-ELSE

The **if-else** node will output one value (T) if the condition (COND) it receives is true, and another (F) if it is False. We want to set up the three **if-else** nodes so that each of the above conditions (less than the lower limit, greater than the upper limit, or neither) returns as different line of text.

Set the T pin of the first **if-else** node to 'too dim' and connect the COND pin to the *less* output. Set the T pin of the second **if-else** node to 'all ok' and connect the COND pin to the **nor** output. Set the T pin of the third **if-else** node to 'too bright' and connect the COND pin to the *greater* output. We will leave the F pins blank, so that nothing is returned when the conditions are false.

**B**

### CONCAT

Use the tab on the variadic **concat** node to expand it to three inputs. Connect all three of the **if-else** outputs to the **concat** inputs. This will combine all three responses into one string. Due to the logic conditions, only one string will be returned at a time and each of the other two nodes will return a blank value. Connect the second **to-bus** node to the **concat** output and label it 'STATE'.

**C**

### INPUT NODES

Connect the **input-port** node to the PORT pin of the **analog-read-no-port-check** node and label it 'PORT'. Connect the **input-pulse** pin to the UPD pin of the **analog-read-no-port-check** node and label it 'UPD'.

**D**

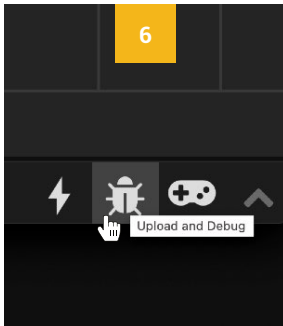
### OUTPUT NODE

Connect a **from-bus** node to the **output-string** node and label both nodes 'STATE'. This may seem redundant, but will help us with our next step.

**E**

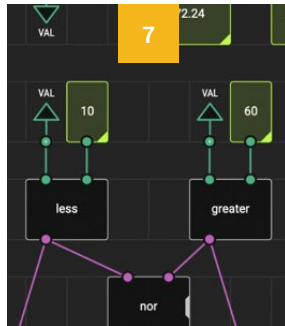
### WATCH NODES

Connect a **from-bus** node to each **watch** node. Label one 'VAL' and one 'STATE'. This will link one to the 100x multiplied output of the sensor node so that you can see the sensor reading, and one to the final output of the **concat** node so that you can see the current state. By adding buses here we can put the two **watch** nodes together and easily view the outputs side by side, rather than having to move around the screen.



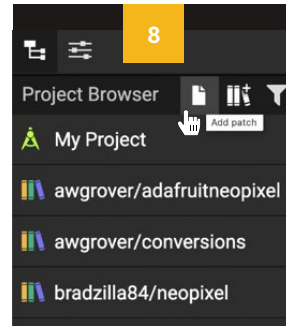
### UPLOAD AND DEBUG

Upload and debug the **light-sensor** patch using the button with the beetle icon.



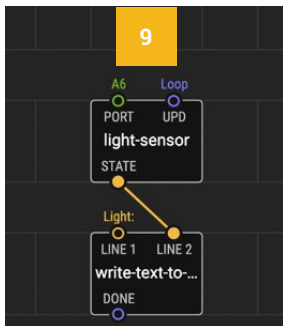
### SET RANGE

Use the **tweak-number** nodes to find a range that works. We have used 10-60 but this may need adjusting to your environment.



### MAKE A NEW PATCH

Add a new patch to the project, and name it '**light-sensor-display**' (or move on to tuto810).



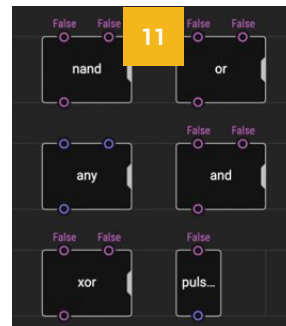
### ADD AND CONNECT NODES

Add your **light-sensor** node and a **write-text-to-oled** node to the patch. Set LINE1 to 'Light:' and connect STATE to LINE2.



### TEST THE PATCH

Upload the patch. Change the light intensity by moving or covering the board. Watch how this affects the screen output.



### EXPERIMENT!

Explore some of the other logic nodes from the **xod/core library**. Or try using **pulse-on-true** with **if-else** to control a programme's timing.



# Sequences and Loops

## Task 9: Requirements

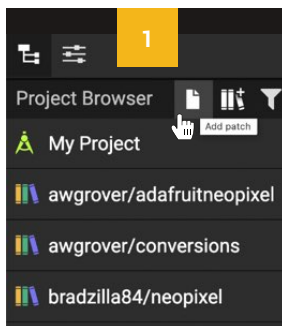
- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (buzzer module)
- USB to micro USB cable

In this final task we'll explore one of the most useful skills for building biological devices: creating sequences and loops.

By programming a sequence of events, using logic and introducing loops we can make devices that are useful for tasks such as automation, monitoring and response to environmental conditions.

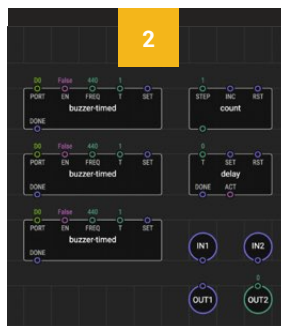
This task will introduce these skills by using the buzzer to play a simple tune 'hot cross buns' (an English nursery rhyme).

This will involve creating two separate sequences and using logic nodes to instruct the programme when to play them. We will use two different methods to make the sequences, and will create a separate node for each one.



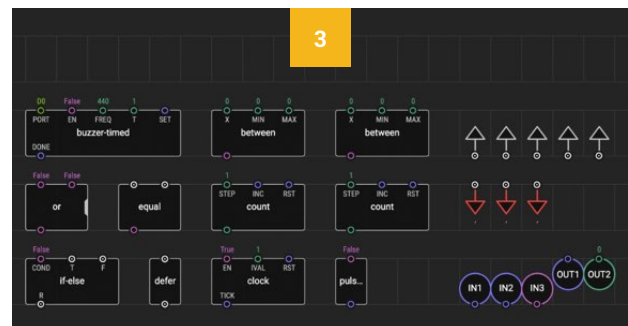
### MAKE THREE NEW PATCHES

Add three new patches to the project, and name them 'bar124', 'bar3' and 'play-tune' (or move on to tuto901).



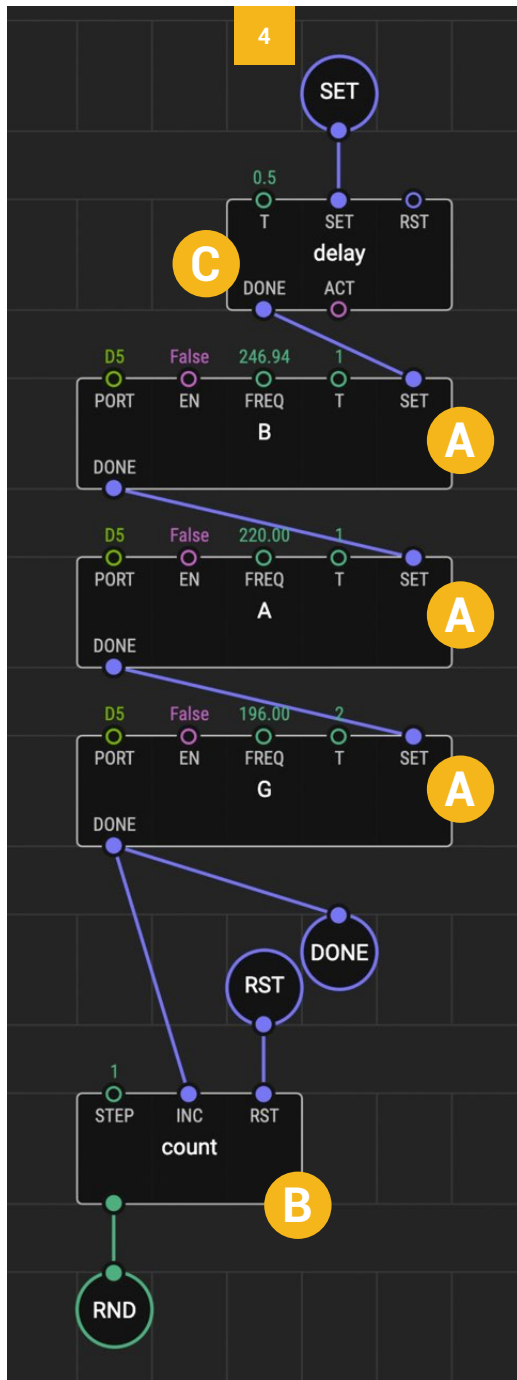
### ADD NODES TO BAR124

Add: **buzzer-timed** x3 (marcoaita/malibrary), **delay**, **count** (xod/core), **input-pulse** x2, **output-pulse**, **output-number** (xod/patch-nodes).



### ADD NODES TO BAR3

Add the following nodes to the **bar3** patch: **buzzer-timed** (marcoaita/malibrary), **clock**, **count** x2, **or**, **defer**, **if-else**, **equal**, **pulse-on-true** (xod/core), **between** x2 (e/comparison - you will need to install this library) **input-pulse** x2, **input-boolean**, **output-pulse**, **output-number**, **to-bus** x3, **from-bus** x5 (xod/patch-nodes).



## SET UP YOUR BAR124 NODE

In this patch we will create the sequence of notes (B, A, G) that make up bars 1, 2 and 4 of the tune.

### A BUZZER-TIMED NODES

Set each of the **buzzer-timed** nodes to play a different note by changing their frequency pins. Set one to '246.94' Hz and change its name to 'B' using the label field in the Inspector. Set one to '220' Hz and name it 'A'. Set one to '196' Hz and name it 'G'. Set PORT to 'D5' and EN to 'False' on all three. Set T (time) to '1' on B and A and to '2' on G. This will produce a longer final note. Connect the DONE pin of B to the SET pin of A and the DONE pin of A to the SET pin of G. Add the *output-pulse* node to the DONE pin of G and name it 'DONE', so that we can see when the sequence is complete.

### B COUNT

Connect the DONE pin of G to the INC pin of **count** so that the count increases each time the sequence is complete. Add the **output-number** node to the output pin and name it 'RND' (round), so that we can see the number of times the sequence has played. Add an **input-pulse** node to the RST (reset) pin and name it 'RST'. We will use this to reset the count when the tune finishes.

### C DELAY

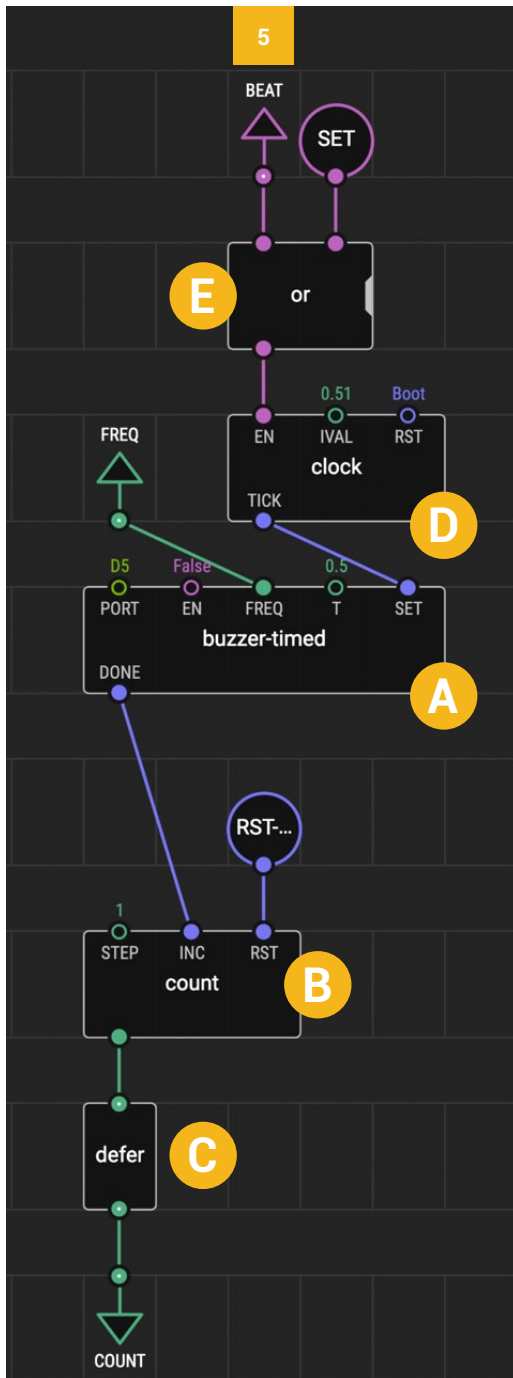
We will use the **delay** node to introduce a half second gap between each bar. Set T (time) to 0.5, and connect the DONE pin of **delay** to the SET pin of B. Add an **input-pulse** node to the SET pin of **delay** and name it 'SET'. We will use this input to initiate the sequence.

To test this node, you can add a **tweak-pulse** node to the SET pin of **delay**. Upload and debug, then use the **tweak-pulse** node to initiate the sequence.

This patch uses a series of nodes with connected SET and DONE pins to create a sequence, similar to how we created our **write-text-to-oled** node in [Task 6](#). The patch for bar 3 will create a sequence in a slightly different way.



# Sequences and Loops



## SET UP YOUR BAR3 NODE (PART 1)

In this patch we will create the sequence of notes (G x 4, A x 4) that makes up bar 3 of the tune.

We will do this in a slightly different way to the **bar134** node. First we will set up a sequence to time the notes, then we will use logic nodes to control the frequency and number of the notes. This will create a loop that feeds information coming out of the programme back into the sequence. We will use buses to connect the two halves of this node.

**A**

### BUZZER-TIMED

Set the port to 'D5' and EN to 'False'. Set T (time) to 0.5. Add a **from-bus** node to the FREQ pin and name it 'FREQ'. This will allow our logic nodes to set the frequency of the note depending on the beat number.

**B**

### COUNT

Connect the **buzzer-timed** DONE pin to the **count** INC pin, so the count increases each time the buzzer sounds. Add an **input-pulse** node to the RST pin and name it 'RST-BEAT'. We will use this to reset the count at the end.

**C**

### DEFER

The **defer** node is the key to creating loops in XOD. In this case, we are creating a loop that reads out the beat number, and changes the frequency of the note and decides whether to repeat based on this. Connect the **count** output pin to the **defer** input pin to inform XOD of this loop. Add a **to-bus** node to the output pin and name it 'COUNT'. This will feed into our logic nodes.

**D**

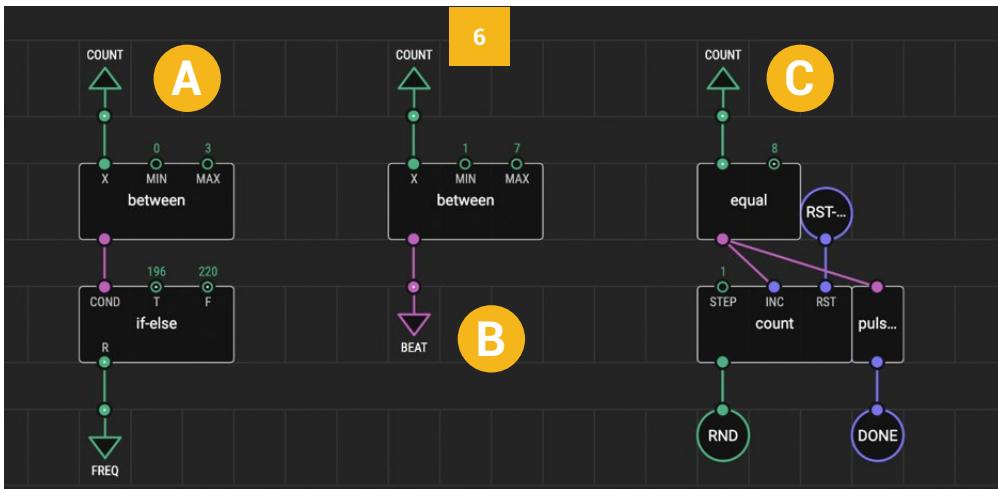
### CLOCK

We want the buzzer to sound a short note repeatedly. To do this, connect the **clock** node to the **buzzer-timed** SET pin. Set RST to 'On Boot' and set the IVAL (interval) pin to 0.51, which is slightly longer than the buzzer sounding time.

**E**

### OR

Using the **clock** node we've made the buzzer sound regularly, but we don't want it to sound all the time. We need to add conditions specifying when the clock is enabled. The buzzer should sound EITHER at the start of the third bar, OR when the sequence has started but not yet finished, i.e. after beats 1-7, but not after beat 8. Connect the **count** output pin to the **defer** input pin to inform XOD of this loop. Add a **to-bus** node to the output pin and name it 'COUNT'. This will feed into our logic nodes.



## SET UP YOUR BAR3 NODE (PART 2)

Follow the instructions below to complete the second half of the loop.

**A**

### FREQUENCY

This part switches the note between G and A depending on the count number. Use one of the *between* nodes. Add a **from-bus** node to the X pin and name it 'COUNT' so that it receives the count number from the end of our sequence. Set MIN to '0' and MAX to '3'. Connect the output to the COND pin of the **if-else** node. Set T to '196' so that the buzzer plays a G for the first four notes (count between 0-3). Set F to '220' so that the buzzer plays an A otherwise. Add a **to-bus** node to the **if-else** output pin and name it 'FREQ'. This will feed back into the loop to set the FREQ pin of the **buzzer-timed** node.

**B**

### BEAT

This part enables the buzzer pulse after beats 1-7. Use the second *between* node. Add a **from-bus** node to X and name it 'COUNT'. Set MIN to '1' and MAX to '7'. Add a **to-bus** node to the output pin and name it 'BEAT'. This will feed back into the loop as one of the conditions that will enable the clock.

**C**

### ROUND

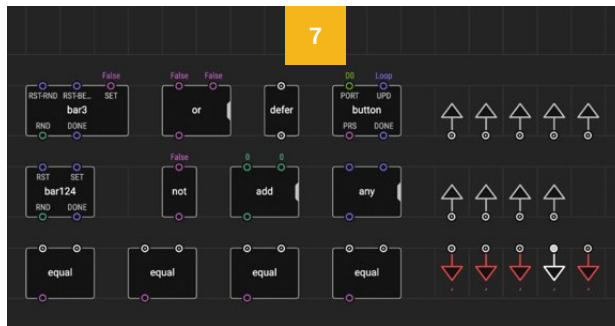
This part records when the sequence is finished. Connect a **from-bus** node to the first input pin of the **equal** node and name it 'COUNT'. Set the second **equal** input pin to '8'. Connect the output pin to the **count** INC pin so that the count increases when the sequence is done. Connect the **input-pulse** node to the **count** RST pin and name it 'RST-RND'. We will use this to reset the count at the end. Add the **output-number** node to the **count** output and name it 'RND' so that we can see the number of times the sequence has played. Connect the **pulse-on-true** node to the **equal** output and then add the **output-pulse** node and name it 'DONE'. This will let us see when the sequence is complete.

Your **bar3** node is now complete. To test this node you can add a **flip-flop** and **tweak-pulse** node to one of the **or** inputs. Pulse this twice to start and stop the sequence.



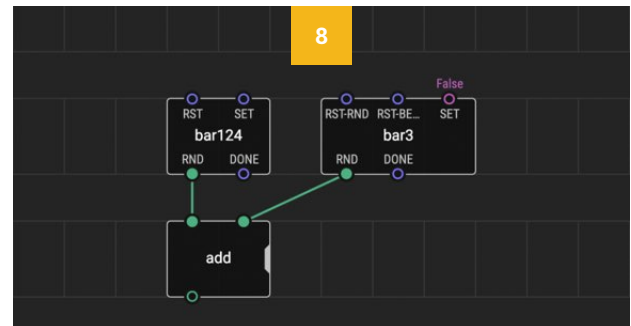


# Sequences and Loops



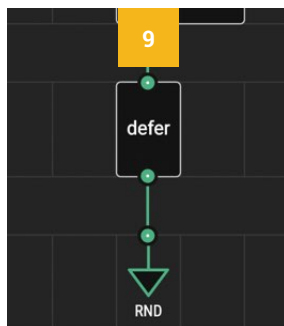
## ADD NODES TO PLAY-TUNE

We will now set up a series of logic conditions to decide when to play each bar. Add the following nodes to the **play-tune** patch: **bar124**, **bar3**, **add**, **defer**, **equal** x4, **or**, **any**, **not** (xod/core), **button** (xod/common-hardware), **to-bus** x5, **from-bus** x9 (xod/patch-nodes).



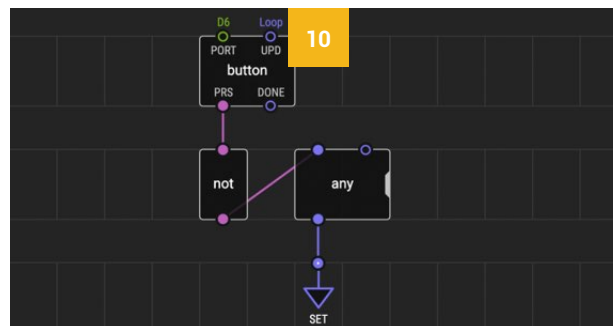
## COUNTING THE ROUND NUMBER

To start we need to know the current round number. We added an **output-number** node (RND) to each of our nodes to count how many times it has been played. Connect both of these RND pins to the **add** node to sum these two values and find the total round number.



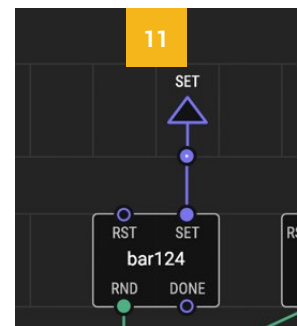
## DEFER AND RND BUS

We will be creating a feedback loop again, so add a **defer** node to the **add** output. Then add a **to-bus** node to this and name it 'RND'.



## PLAYING BAR 1

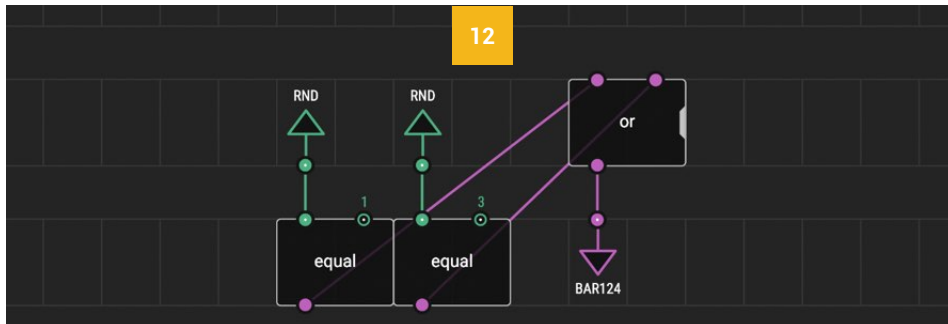
We will use the button on the board to initiate round 1. Set the **button** PORT pin to 'D6' and PRS pin to 'True'. Then connect the PRS pin to the **not** node. Connect **not** output to one of the inputs of **any**. Leave the other **any** input unconnected for now. Add a **to-bus** node to the **not** output pin and name it 'SET'.



## SET BUS

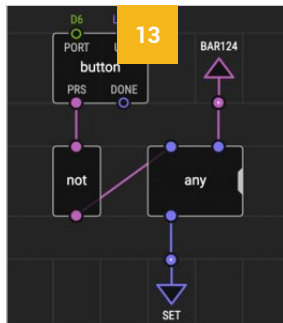
Add a **from-bus** node to the SET pin of **bar124** and name it 'SET'. This will initiate the **bar124** sequence.





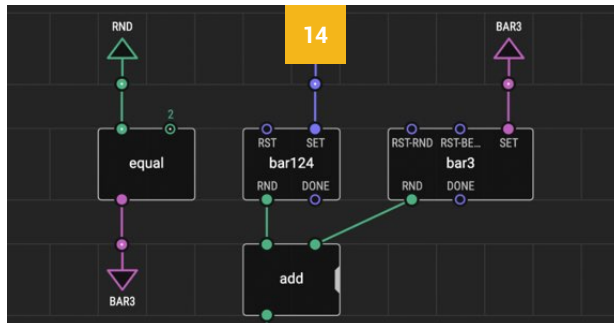
### PLAYING BARS 2 AND 4

We also want to initiate **bar124** at the end of bars 1 and 3. Use two **equal** nodes. Connect a **from-bus** node to the first input of each and name them both 'RND' so that they receive the round number. Set the second input of one to '1' and the other to '3'. Connect the outputs of both nodes to the input pins of the **or** node. Add a **to-bus** node to the output of **or** and name it 'BAR124'.



### BAR124 BUS

Add a **from-bus** node to the other **any** input (see [Step 10](#)). Name it 'BAR124'. This will set the **bar124** sequence after bars 1 and 3.

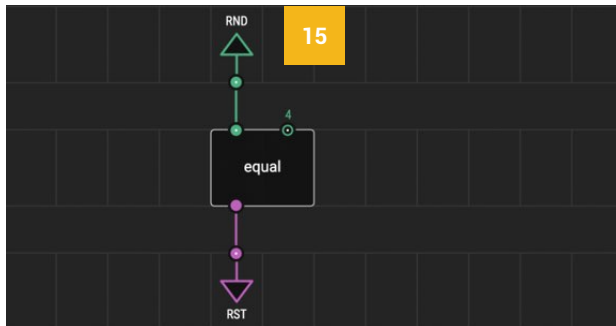


### PLAYING BAR 3 AND BAR3 BUS

We want **bar3** to play at the end of bar 2. Add a **from-bus** node to the input of the third **equal** node and name it 'RND'. Set the second input of **equal** to 2. Add a **to-bus** node to the output and name it 'BAR3'. Add a **from-bus** node to the SET pin of **bar3** and name it 'BAR3'. This bus will initiate **bar3** after bar 2.

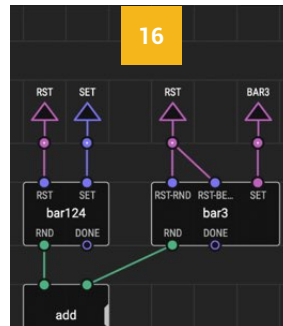


# Sequences and Loops



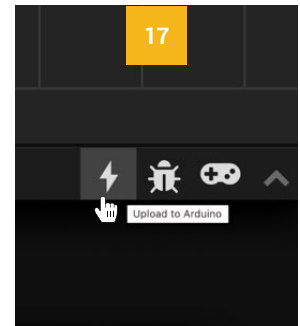
## RESETTING THE SEQUENCE

After round 4 we want to reset the sequence so that the round count returns to 0. We added RST pins to our **bar124** and **bar3** nodes for this. Add a **from-bus** node to the first input of the final **equal** node and name it 'RND'. Set the second input to '4' so the reset happens after bar 4. Add a **to-bus** node to the output pin and name it 'RST'.



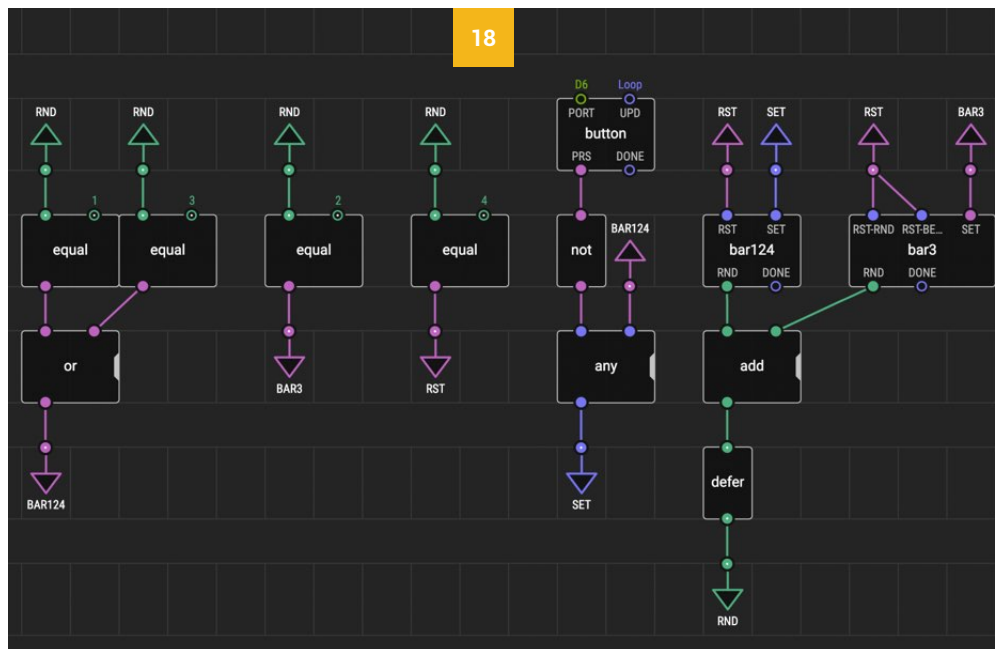
## RST BUS

Name both **from-bus** nodes 'RST'. Add one to the RST pin of **bar124** and one to the RST pins (RST-RND, RST-BEAT) of **bar3**.



## UPLOAD AND TEST

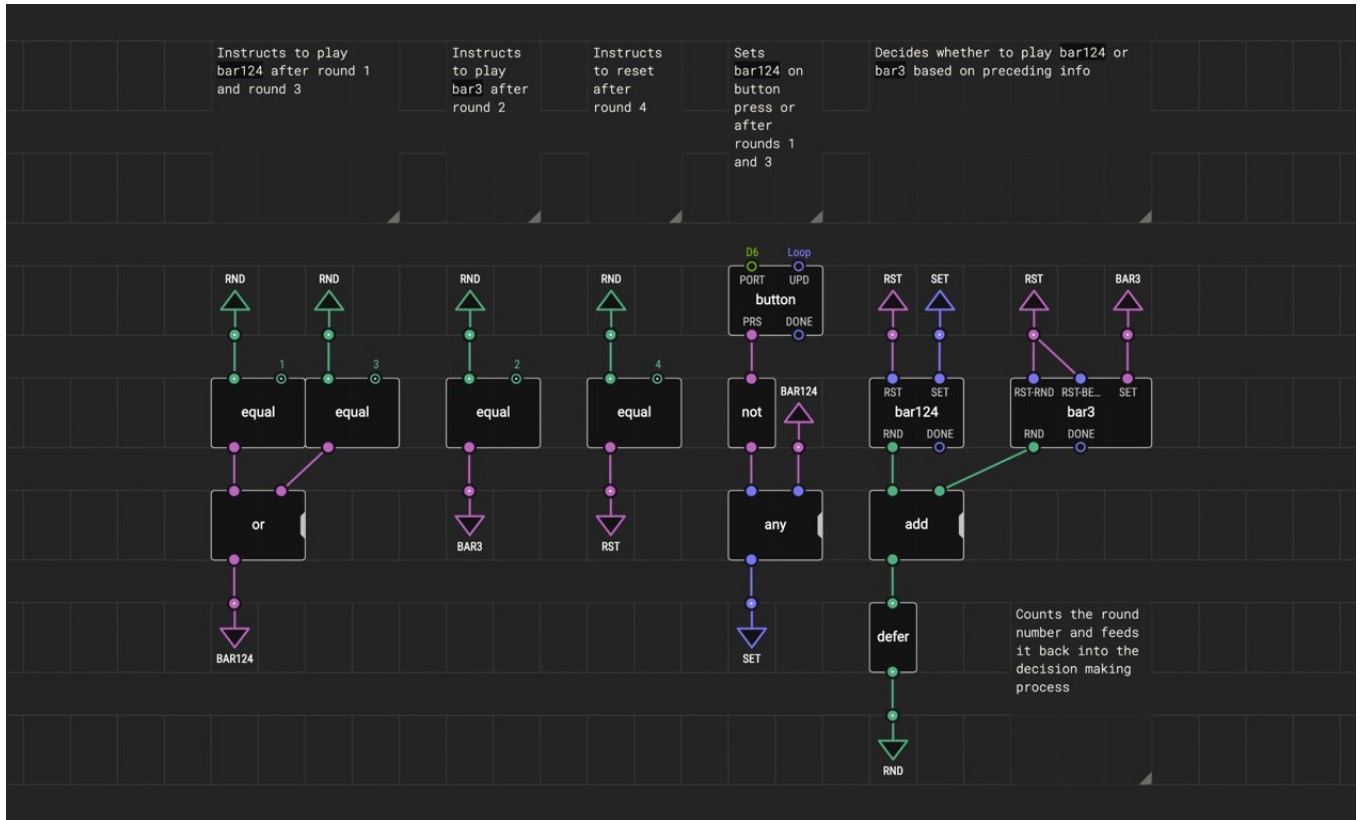
Finally, upload the programme and test it out by pressing the button on your board!



## EXPERIMENT!

Congratulations, you've completed the final task! Your final patch should look something like this.

Play around with this patch and the nodes you have used. Try creating a different tune. Or try to make the LED flash along in time with the notes. The possibilities are endless now that you understand the core principles.



## Comment Boxes

When creating a more complex programme like this it is often useful to include comment boxes, both to help keep track of what you are doing, and to make it easier for others to follow your workflow. This is the XOD equivalent of 'commenting out' notes when writing code.

In the example above you can see that comments have been added above each section of the programme to describe what that part of the patch is doing. Try adding your comments to annotate your **play-tune** patch. You can add a comment box by navigating to 'Edit > Insert Comment' in the menu bar.

You can also add formatting to your XOD comment boxes, for example:

- \*Surround text with stars to add bold white text\*
- \*\*Surround text with two stars to add bold red text\*\*
- - Use a dash before text to add a bullet list
- 1. Use a number and point before text to add a number list

You can read more about adding comments to document your nodes and XOD 'markdown' (formatting) on the XOD website at [www.xod.io/docs/guide/documenting-nodes](http://www.xod.io/docs/guide/documenting-nodes).



# Embedded hardware

## 1 LED

Node: `xod/common-hardware/led`  
 Settings: PORT = D4  
 LUM = luminance (brightness) between 0-1  
 ACT = True  
 Used in: [Task 1](#), [Task 4](#)

## 2 BUZZER

Node: `marcoaita/malibrary/buzzer`  
 Settings: PORT = D5  
 EN = True  
 FREQ = 440  
 Used in: [Task 2](#), [Task 9](#)

## 3 OLED SCREEN (SSD1306)

Nodes: `wayland/ssd1306-oled-i2c/ssd1306-oled-i2c-device`  
`wayland/ssd1306-oled-i2c/clear-display`  
`wayland/ssd1306-oled-i2c/send-buffer-to-display`  
 Settings: ADDRESS = 3Ch  
 WEIGHT = 128  
 HEIGHT = 64  
 RESET = -1  
 Notes: Add any other nodes from `wayland/ssd1306-oled-i2c` between `clear-display` and `send-buffer-to-display`. Connect `ssd1306-oled-i2c-device` DEV to all DEV pins. Connect `clock` node to `clear-display` UPD then connect each UPD pin in turn.  
 Used in: [Task 6](#), [Task 7](#), [Task 8](#)

## 4 BUTTON

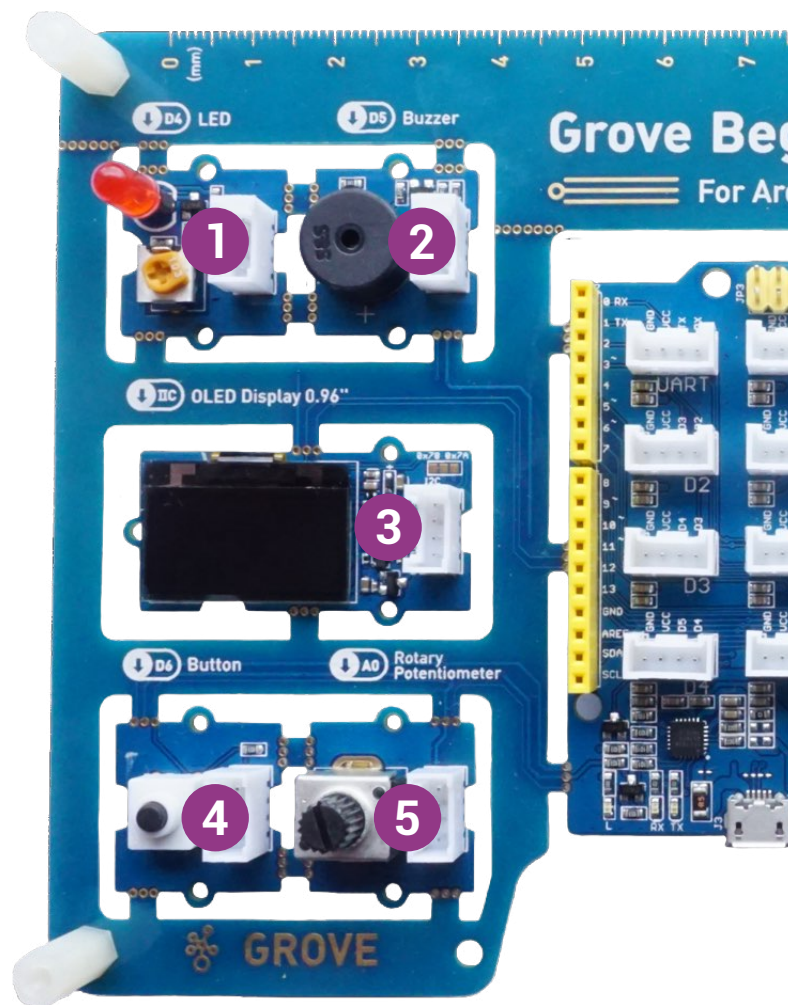
Node: `xod/common-hardware/button`  
 Settings: PORT = D6  
 UPD = Loop  
 Notes: `button` is automatically on and turns off with a press. Use a `not` node to invert this.  
 Used in: [Task 2](#)

## 5 ROTARY POTENTIOMETER

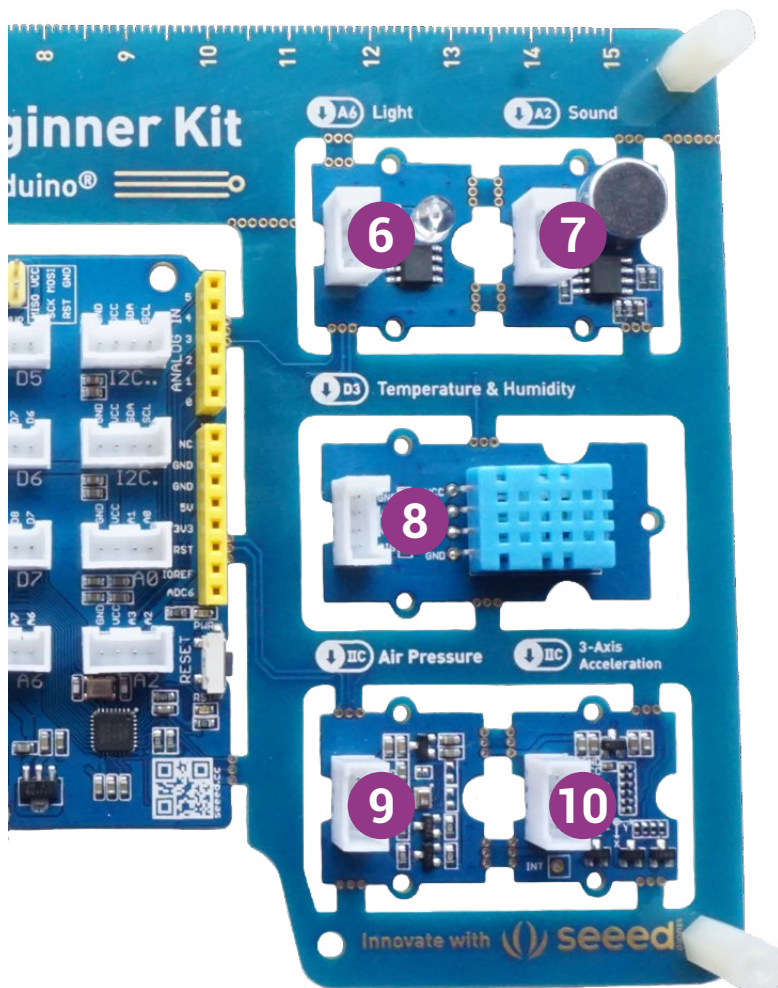
Node: `xod/common-hardware/pot`  
 Settings: PORT = A0  
 UPD = Loop  
 Used in: [Task 2](#)

## Grove Board Cheat-Sheet

This cheat-sheet provides a quick guide to which XOD node or library to use for each of the inbuilt components on the Grove All-In-One Beginner Kit for Arduino.



The guide suggests a node for each component as well as some standard settings. Other nodes and settings can also be used, and we encourage play with node settings and trying new nodes where available.



## 6 LIGHT SENSOR

Node: `wayland/analog-read-no-port-check/`  
`analog-read-no-port-check`

Settings: PORT = A6  
UPD = Loop

Used in: [Task 8](#)

## 7 SOUND SENSOR

Node: `xod/common-hardware/analog-sensor`

Settings: PORT = A2  
UPD = Loop

Used in: [Task 6](#)

## 8 TEMPERATURE AND HUMIDITY SENSOR (DHT11)

Node: `xod-dev/dht/dht11-hygrometer`

Settings: PORT = D3  
UPD = Connect *clock* node

Notes: Setting UPD to 'Loop' can cause errors.

Used in: [Task 3](#)

## 9 AIR PRESSURE SENSOR (BMP280)

Node: `wayland/bmp280-barometer/barometer-thermometer`

Settings: MODE = 03h  
OST = 02h  
OSP = 05h  
FILT = 04h  
STDBY = 04h  
UPD = Loop

Used in: [Task 5](#)

## 10 3-AXIS ACCELERATION SENSOR (LIS3DH)

Node: `wayland/lis3dh-accelerometer/accelerometer`

Settings: ADDR = 19h  
RATE = 07h  
RANGE = 00h

Used in: [Task 7](#)



# XOD nodes for the embedded hardware

The Grove Beginner Kit consists of a central microcontroller on an Arduino-compatible board linked to a series of ten different components on surrounding parts of the circuit board. Within the intact board, copper traces allow the microcontroller to communicate with the components via ports indicated above. (If needed, the individual component daughter boards can be cut from the host, and used as individual modules, wired via the white-coloured Grove connectors. "Grove" refers to the plug standard used for interconnection of the different digital, analogue, SPI, serial and I2C connections).

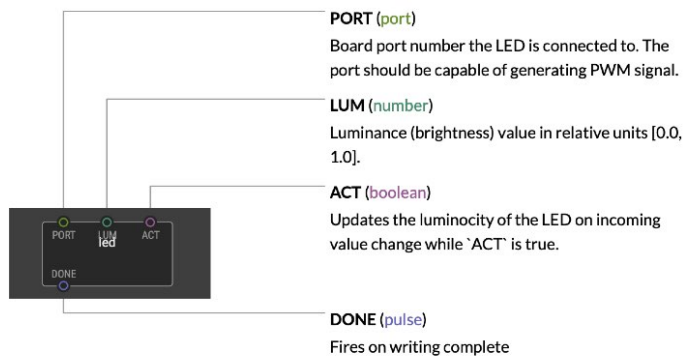
Different XOD nodes from the standard XOD installation, and from external libraries, can be used to control the different peripheral components. These are shown below:

## 1 LED

Dimmable light emitting diode (LED). Is connected to a pulse-width modulated (PWM) digital port (D4). Controlled by the `xod/common-hardware/led` node.

### `xod/common-hardware/led`

Drives a generic single color dimmable LED. The conversion between luminance and LED power is biased so that change in brightness on range 0...1 is perceived more or less uniformly by a human. Possible errors: — Invalid port

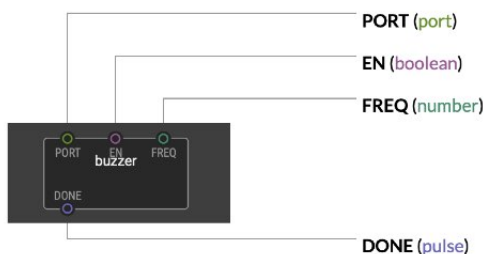


## 2 Buzzer

The piezo can be connected to digital outputs and will emit a tone when the output is high. Alternatively, it can be connected to an analog pulse-width modulation output to generate various tones and effects.

### `marcoaita/malibrary/buzzer`

A wrapper around the tone command for Arduino. It allows to use the buzzer on the Ks0183 keystudio Multi-purpose Shield V1



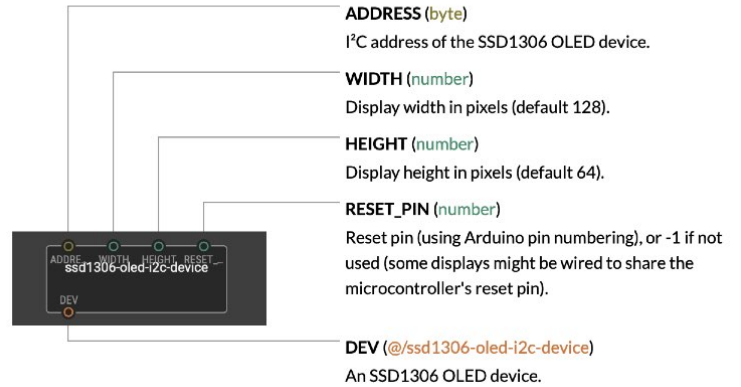


### 3 OLED screen

The OLED Display 0.96" (SSD1306) is a monochrome(white) 128x64 pixels display matrix module with I2C Interface.

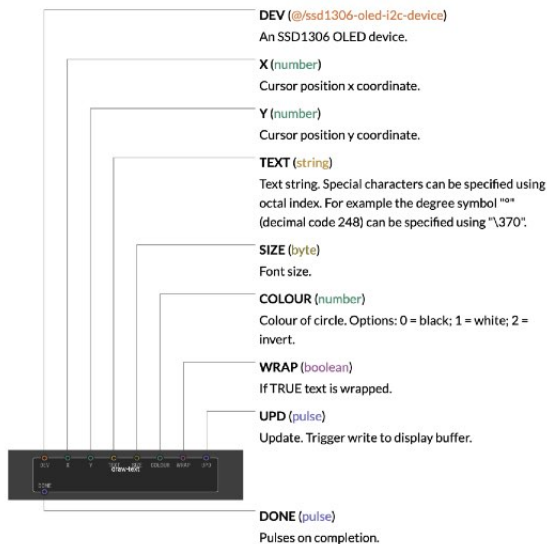
[wayland/ssd1306-oled-i2c/ssd1306-oled-i2c-device](#)

Create SSD1306 OLED device.



[wayland/ssd1306-oled-i2c/draw-text](#)

Writes string to display buffer. To show content of display buffer on screen use node send-buffer-to-display.



### External XOD library

[wayland/ssd1306-oled-i2c@0.0.7](#)

Arduino library for OLED displays driven by the SSD1306 chip.

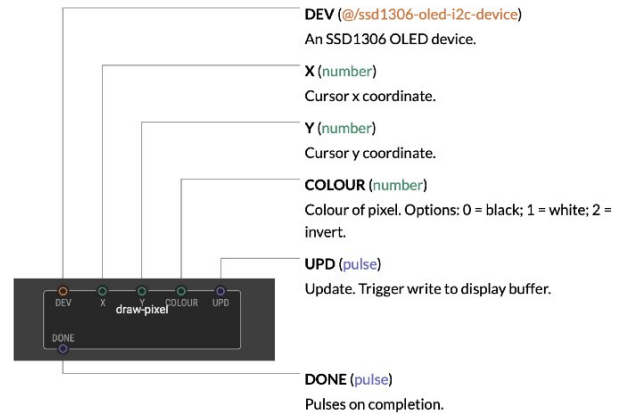
Communication via I2C. Wraps Adafruit\_SSD1306 ([https://github.com/adafruit/Adafruit\\_SSD1306](https://github.com/adafruit/Adafruit_SSD1306)).

- **clear-display:** Clear contents of display buffer (set all pixels to off). Changes buffer contents only, no immediate effect on display. Follow up with a call to send-buffer-to-display, or with other graphics commands as needed by one's own application.
- **dim-display:** Dim display. This has an immediate effect on the display, no need to use the send-buffer-to-display node -- buffer contents are not changed.
- **draw-circle:** Draw a circle. Writes data to display buffer. To show content of display buffer on screen use node send-buffer-to-display.
- **draw-line:** Draw a line. Data written to display buffer. To show content of display buffer on screen use node send-buffer-to-display.
- **draw-pixel:** Draw a pixel.
- **draw-rectangle:** Draw a rectangle. Writes data to display buffer. To show content of display buffer on screen use node send-buffer-to-display.
- **draw-rounded-rectangle:** Draw a rounded rectangle. Writes data to display buffer. To show content of display buffer on screen use node send-buffer-to-display.
- **draw-text:** Writes string to display buffer. To show content of display buffer on screen use node send-buffer-to-display.
- **draw-triangle:** Draw a triangle. Writes data to display buffer. To show content of display buffer on screen use node send-buffer-to-display.
- **draw-xod-logo:** Draw XOD logo. Logo is 128 x 64 pixels.
- **example-draw-shapes:** Draw shapes in proportion to display size.

- **example-rotate-display:** Text will be drawn in two different directions.
- **example-scroll:** Performs a diagonal right scroll for 30 seconds, then stops scrolling.
- **example-xod-logo:** Display and inversion of XOD logo.
- **get-display-dimensions:** Get dimensions of display in pixels.
- **invert-display:** Enable or disable display invert mode (white-on-black vs black-on-white). This has an immediate effect on the display, no need to use the send-buffer-to-display node -- buffer contents are not changed, rather a different pixel mode of the display hardware is used. When enabled, drawing BLACK (value 0) pixels will actually draw white, WHITE (value 1) will draw black.
- **rotate-display:** No description
- **send-buffer-to-display:** Push data currently in RAM to SSD1306 display. Drawing operations are not visible until this function is called. Call after each graphics command, or after a whole set of graphics commands, as best needed by one's own application.
- **ssd1306-oled-i2c-device:** Create SSD1306 OLED device.
- **start-scroll:** Activate scroll for all or part of the display. To scroll whole display set FIRST to 00h and LAST to 0Fh.
- **stop-scroll:** Stop scrolling.

wayland/ssd1306-oled-i2c/draw-pixel

Draw a pixel.

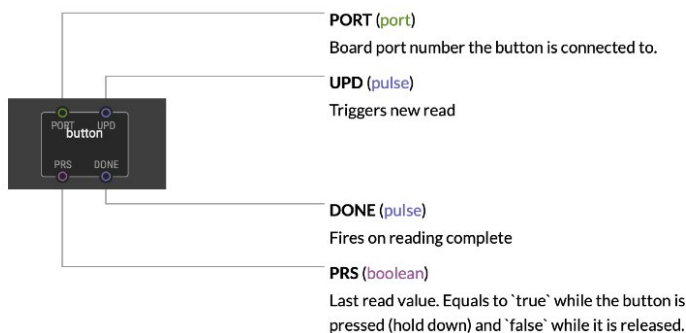


## 4 Button

The Grove Button is a momentary push button. It contains one independent "momentary on/off" button. "Momentary" means that the button rebounds on its own after it is released. The button outputs a HIGH signal when pressed, and LOW when released.

xod/common-hardware/button

Reads a generic button or another mechanical switch. It is expected that the button is low while pressed, the on-board pull-up resistor is enabled if possible. The node provides signal debounce with 20 ms settle delay. Possible errors: – Invalid port





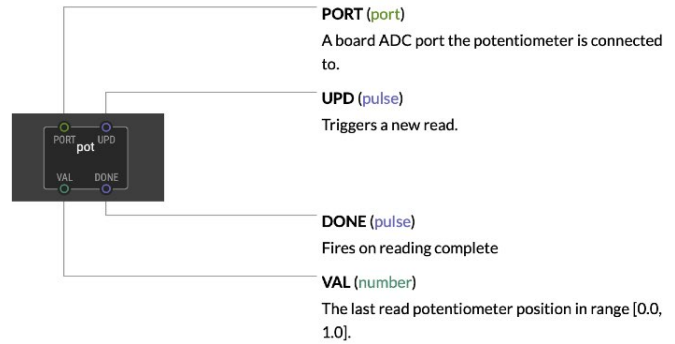
## 5 Potentiometer

The potentiometer or pot, consists of an internal resistive element called the track and a sliding contact called the wiper where end terminals are attached to the resistive element. These allow the output of a variable voltage that is converted to an output value of 0 to 1.



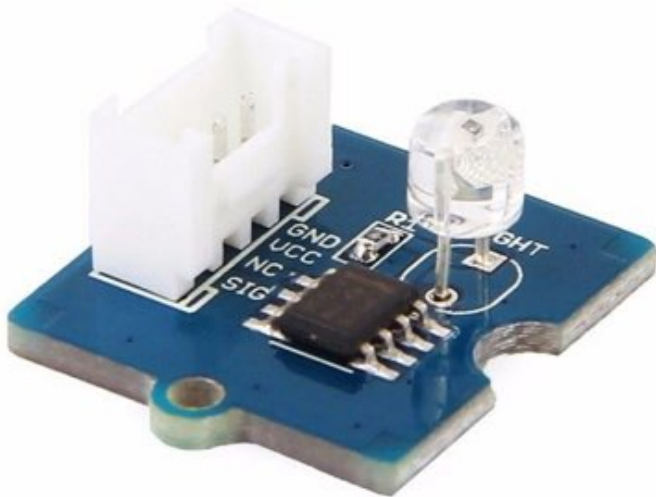
`xod/common-hardware/pot`

Reads values from a generic potentiometer. Basically a thin wrapper around 'analog-input' for a simpler learning experience. Possible errors: — Invalid port



## 6 Light sensor

The Grove - Light sensor integrates an LS06-S photo-resistor (light dependent resistor) to detect the intensity of light. The resistance of photo-resistor decreases when the intensity of light increases. A dual op-amp chip LM358 on board produces voltage corresponding to the intensity of light (i.e. based on resistance value). The output signal is analog value, the brighter the light is, the larger the value, with a short response time: 20 ~ 30 milliseconds.



`wayland/analog-read-no-port-check/analog-read-no-port-check`

Reads analog signal value from board ADC port. Does not check if port is valid.



### External XOD library

`wayland/analog-read-no-port-check`

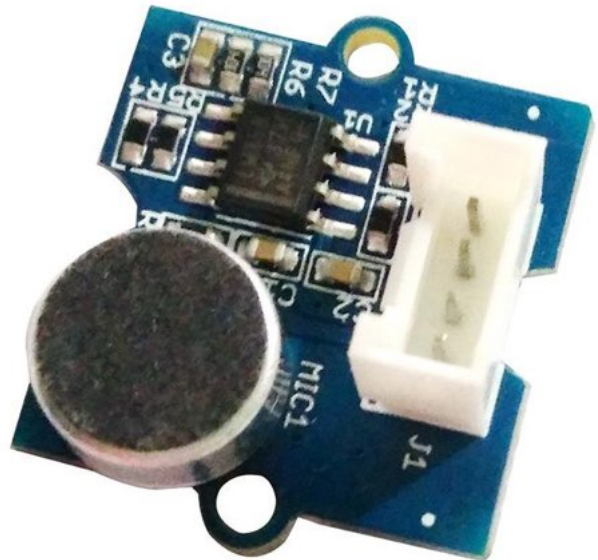
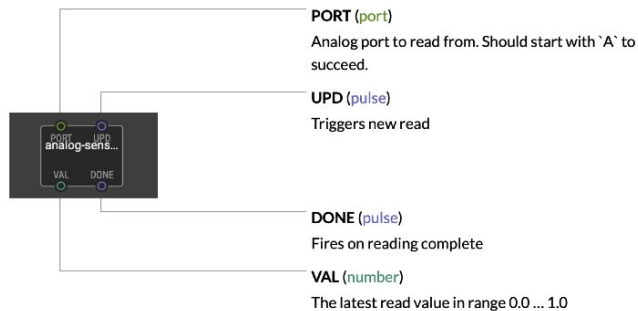
The device is connected to analog port 6 on the Beginner board. This is usually not connected on Arduino UNO compatible boards, so you should use a custom **analog read** node that has been written by Matt Wayland (<https://xod.io/libs/wayland/analog-read-no-port-check/>). This allows the output voltage of the sensor to be converted to a digital signal by the analog-to-digital-converter on your controller board.

## 7 Sound sensor

Grove - Sound Sensor can detect the sound intensity of the environment. The main component of the module is a simple microphone, which is based on the L358 amplifier and an electret microphone. This module's output is analog and can be easily sampled and tested by an Arduino microcontroller.

[xod/common-hardware/analog-sensor](#)

Reads analog signal value from analog sensor. Possible errors: — Invalid port



## 8 DHT11 Hygrometer (analog)

(Please note that Seeed Studio supply different version of the Grove Beginner's board with either the DHT11 (blue coloured) or the DHT20 (black coloured). These devices are similar in function, but require different nodes and communication routines!)

Grove - DHT11 Temperature & Humidity Sensor is a high quality, low-cost digital temperature, and humidity sensor based on the DHT11 module. DHT11 is the most common temperature and humidity module for Arduino and Raspberry Pi. It is widely favored by hardware enthusiasts for its many advantages such as low power consumption and excellent long-term stability. Relatively high measurement accuracy can be obtained at a very low cost. The single-bus digital signal is output through the built-in ADC, which saves the I/O resources of the control board.

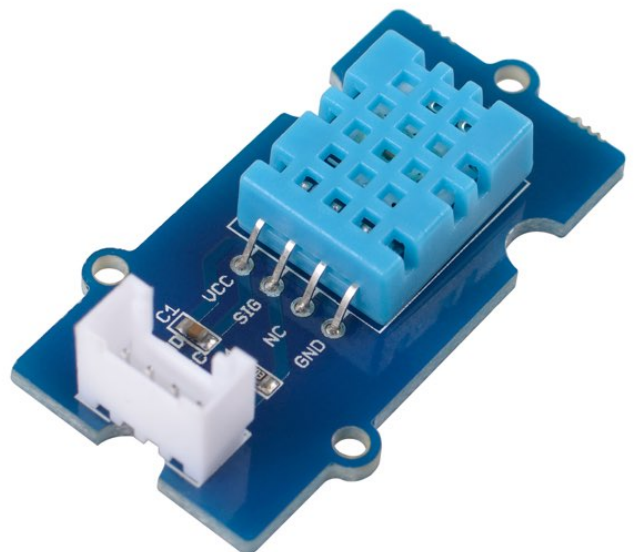
The Grove - Temperature & Humidity Sensor uses an upgraded version of DHT11. The new version of the DHT11 module replaces resistive humidity components with capacitive humidity components. The temperature and humidity measurement range are wider. The temperature resolution is higher.

### External XOD library

[xod-dev/dht@0.36.1](#)

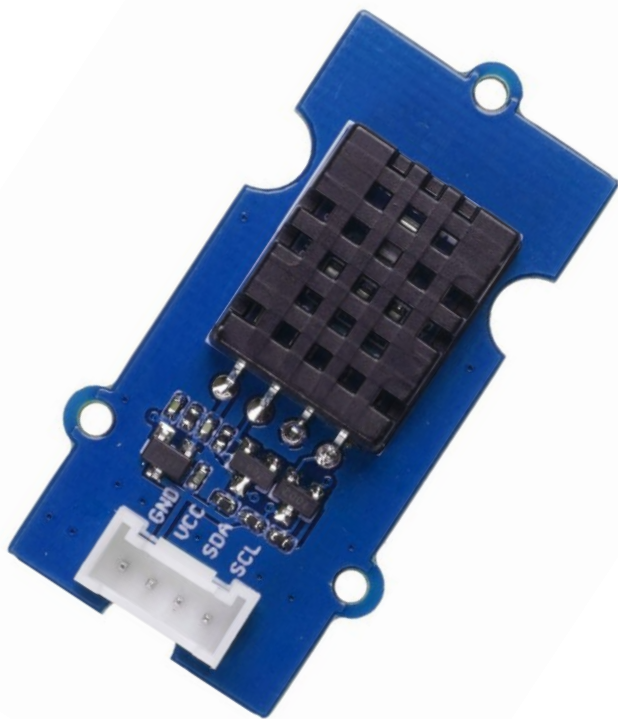
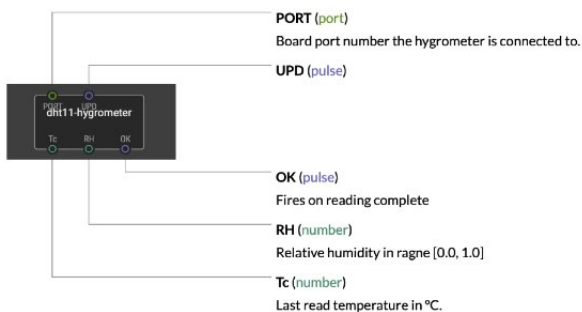
Nodes to work with DHT11 or DHT21 sensors, or compatible sensors: RHT01, DHT22, DHT33, DHT44, AM2301, HM2301, AM2302, AM2303, RHT02, RHT03, RHT04, RHT05.

- **dht11-device**: Represents a DHT11 sensor. Also named RHT01.
- **dht11-hygrometer**: Read the temperature and humidity by the DHT11 (RHT01) hygrometer sensor.
- **dht2x-device**: Represents a DHT21 or compatible sensor: DHT21, DHT22, DHT33, DHT44, AM2301, HM2301, AM2302, AM2303, RHT02, RHT03, RHT04, RHT05.
- **dht2x-hygrometer**: Read the temperature and humidity by the DHT21 or compatible (DHT21, DHT22, DHT33, DHT44, AM2301, HM2301, AM2302, AM2303, RHT02, RHT03, RHT04, RHT05) hygrometer sensor.



**xod-dev/dht/dht11-hygrometer**

Read the temperature and humidity by the DHT11 (RHT01) hygrometer sensor.

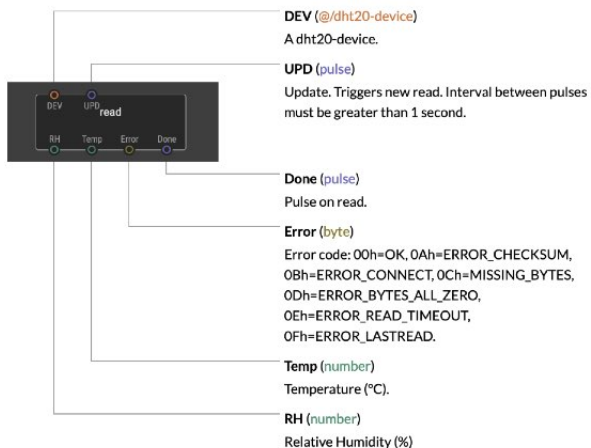


- **example-dht11:** No description
- **read:** Reads the temperature and humidity.
- **read(dht11-device):** Reads the temperature and humidity.
- **read(dht2x-device):** Reads the temperature and humidity.

	DHT22	DHT20	DHT11
<b>Supply Voltage</b>	DC: 3.0 - 6.0V	DC: 2.0 - 5.5V	DC: 3.3 - 5.5V
<b>Measuring Range (Temperature)</b>	-40 ~ + 80 °C	-40 ~ + 80 °C	-20 ~ + 60 °C
<b>Measuring Range (Humidity)</b>	0 ~ 100% RH	0 ~ 100% RH	5 ~ 95% RH
<b>Temperature Accuracy</b>	± 0.5 °C	± 0.5 °C	± 1 °C
<b>Humidity Accuracy</b>	± 2 % RH ( 25 °C )	± 3 % RH ( 25 °C )	± 5 % RH ( 25 °C )
<b>Output Signal</b>	I2C signal	I2C signal	1-Wire

**wayland/dht20/read**

Measure relative humidity and temperature.



**DHT20 Hygrometer (I2C)**

The newer Grove - Temperature & Humidity Sensor is based on the DHT20 sensor. The DHT20 is an upgraded version of the DHT11, compared with the previous version, the temperature and humidity measurement accuracy are higher, and the measurement range is larger. It features I2C output.

**External XOD library wayland/dht20@0.0.3**

from: ASAIR DHT20 humidity and temperature sensor ([https://cdn-shop.adafruit.com/product-files/5183/5193\\_DHT20.pdf](https://cdn-shop.adafruit.com/product-files/5183/5193_DHT20.pdf)). Wraps <https://github.com/RobTillaart/DHT20>

- **init:** Initialize dht20-device.
- **dht20-device:** Create a dht20-device.
- **read:** Measure relative humidity and temperature.
- **example:** Patch for testing sensor. Run in debug mode.
- **hygrometer-thermometer:** Combines lower level nodes to create a ready to use sensor.



## 9 Air pressure sensor

The Grove BMP280 Barometer Sensor is built around Bosch BMP280, it is a low-cost and high-precision environmental sensor that measures the temperature and air pressure. This sensor supports both I2C and SPI communication using a custom BMP280 Arduino library.

Grove BMP280 provides precise measurements of barometric pressure and temperature in the environment. The air pressure can be measured in a range from 300 hPa to 1100hPa with  $\pm 1.0$  hPa absolute accuracy. It also provides a temperature readout, for temperatures between  $-40^{\circ}\text{C}$  and  $85^{\circ}\text{C}$  with an accuracy of  $\pm 1^{\circ}\text{C}$ .

Owing to its high accuracy in measuring air pressure, and known pressure changes with altitude, one can calculate the altitude with  $\pm 1$  meter accuracy, which makes it a precise altimeter as well. It provides both I2C and SPI interfaces for communication with the microcontroller. The board provides alternative I2C addresses.

### External XOD library

#### wayland/bmp280-barometer@0.0.1

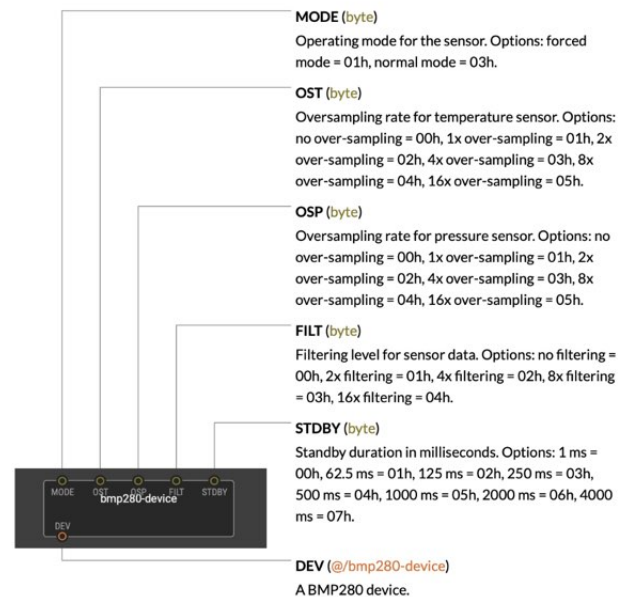
Library authored by Matt Wayland for the BMP280 barometric pressure and temperature sensor. Converted from [https://github.com/adafruit/Adafruit\\_BMP280\\_Library](https://github.com/adafruit/Adafruit_BMP280_Library).

- **bmp280-device**: Create BMP280 device. See datasheet ([https://ae-bst.resource.bosch.com/media/\\_tech/media/datasheets/BST-BMP280-DS001.pdf](https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001.pdf)) for recommended oversampling and filter settings for specific use cases.
- **calculate-altitude**: Calculate altitude from atmospheric pressure.
- **read-pressure**: Read pressure in Pascal.
- **read-temperature**: Read temperature in degrees Celsius.
- **example-oled-altimeter**: Uses bmp280 as an altimeter and displays altitude on an OLED screen.
- **barometer-thermometer**: Combines low level nodes to create a simple to use barometer and thermometer.
- **example-test-barometer-thermometer**: Patch to test barometer-thermometer node. Run in debugger.



#### wayland/bmp280-barometer/bmp280-device

Create BMP280 device. See datasheet ([https://ae-bst.resource.bosch.com/media/\\_tech/media/datasheets/BST-BMP280-DS001.pdf](https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001.pdf)) for recommended oversampling and filter settings for specific use cases.



## 10 Accelerometer

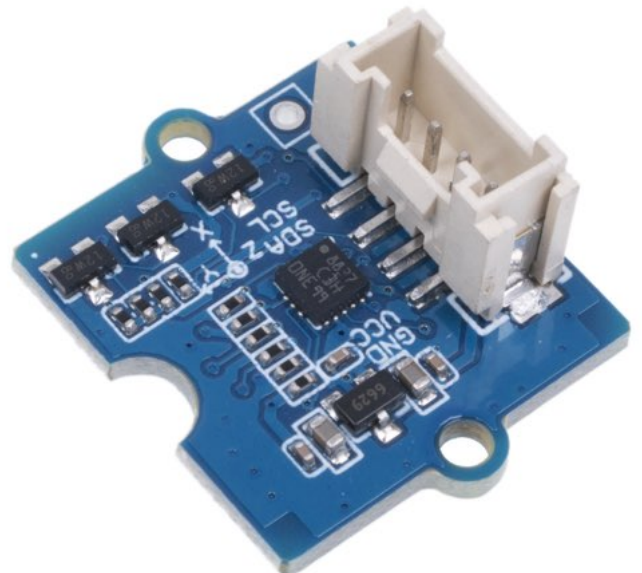
The Grove 3-Axis Digital Accelerometer contains a low-cost 3 axis accelerometer (LIS3DHTR). It is based on the LIS3DHTR chip which provides multiple measurement ranges:  $\pm 2g$ ,  $\pm 4g$ ,  $\pm 8g$ ,  $\pm 16g$ . The tiny 3 - Axis accelerometer can support I2C, SPI, and ADC GPIO interfaces, which means you can choose any way to connect with your development board. In addition, this accelerometer can also be temperature compensated to tune the errors.

### External XOD library

#### wayland/lis3dh-accelerometer@0.0.1

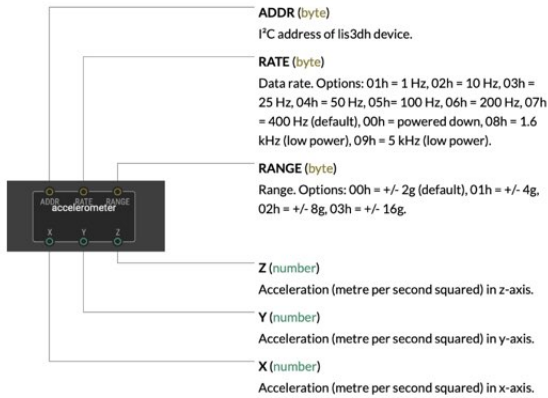
Authored by Matt Wayland to support the LIS3DH triaxial accelerometer. Wraps [https://github.com/adafruit/Adafruit\\_LIS3DH](https://github.com/adafruit/Adafruit_LIS3DH).

- **accelerometer**: Combines lower level nodes to create a simple to use accelerometer.



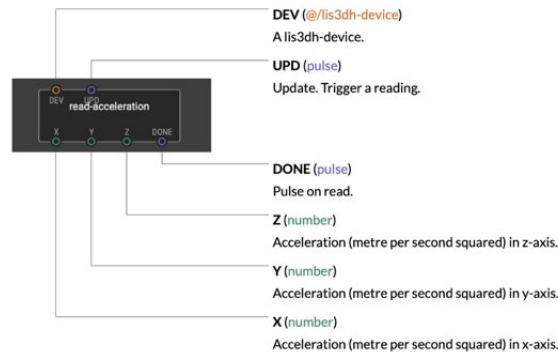
wayland/lis3dh-accelerometer/accelerometer

Combines lower level nodes to create a simple to use accelerometer.



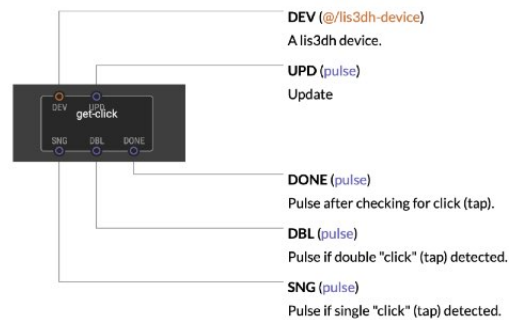
wayland/lis3dh-accelerometer/read-acceleration

Read acceleration (metre per second squared) in all three axes.



wayland/lis3dh-accelerometer/get-click

Detect single or double "click" (tap).



- **click-detector:** Combines low level nodes to create a simple to use click-detector. Detects single or double taps of the sensor.
- **example-read-adc:** Patch to test read-adc node. Run in debugger.
- **example-read-raw:** Patch to test read-raw node. Run in debugger.
- **get-click:** Detect single or double "click" (tap).
- **get-data-rate:** Get data rate.
- **get-device-id:** Read the ID of the lis3dh device.
- **get-range:** Read the g range for the accelerometer.
- **lis3dh-device:** Create a lis3dh device.
- **read-acceleration:** Read acceleration (metre per second squared) in all three axes.
- **read-adc:** Read the auxiliary analog-to-digital converter.
- **read-raw:** Read X, Y and Z raw values.
- **set-click:** Configure parameters for "click" (tap) detection. See datasheet for explanation of parameters: [http://www.st.com/resource/en/application\\_note/cd00290365.pdf](http://www.st.com/resource/en/application_note/cd00290365.pdf)
- **set-data-rate:** Set data rate.
- **set-range:** Set range.
- **example-motion-detector:** Demonstrates how lis3dh can be used as a motion sensor. LED is illuminated if motion is detected. Push button resets motion detector.
- **example-test-accelerometer:** Test of accelerometer. Run in debugger.
- **example-test-click-detector:** Test of click-detector. Run in debugger.
- **example-tweak-settings:** Demonstrates changing data-rate and range at runtime using tweak nodes. Run in debugger.



# Lesson 5: Next Steps



**Hardware expansion**



**Case studies**



**Additional information**

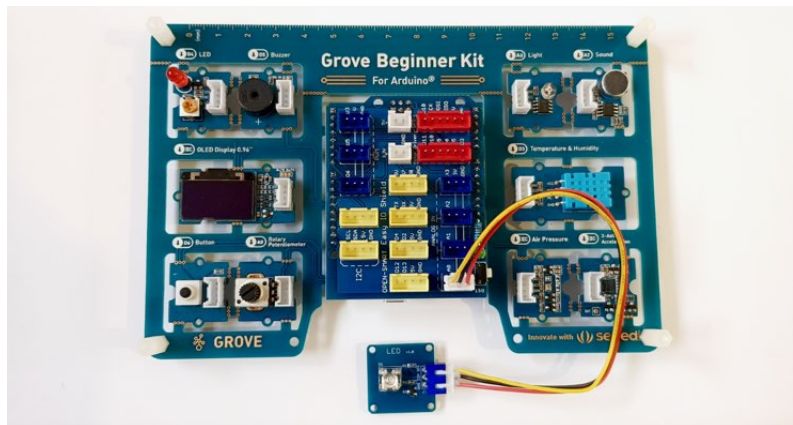


# Next steps

Congratulations, you have completed the first part of the No-Code Programming for Biology beginner's course! Starting from understanding your board and how to programme it, through to building your own nodes and complex sequences, this guide has taken you through how to use each of the onboard devices, as well as how to preform a range of useful functions in XOD.

You should now be comfortable with your board and the XOD software, and should have gained a better understanding of how these tools can be used to create programmes and devices that respond to, and influence, their environment. Once you understand the basic principles of programming microcontrollers such as this, the possibilities for applications are endless.

In the remaining part of this book, we will provide information about how you can build on these skills to start developing your own custom devices, as well as where to find components, information and help to guide you in your next steps. We will also provide some examples of how previous Biomaker participants have applied these skills to real-world applications to assist with biological research in the lab and field. Finally we provide some additional useful information including an overview of alternative development boards, a list of useful websites, a Grove board cheat sheet, a list of XOD nodes used in this guide, and a glossary of terms.



*Open Smart easy-plug LED breakout board connected to the Grove board*

## OBJECTIVES

By the end of this chapter you should be able to:

- Recall where to find additional components compatible with your board.
- Recognise the different ways to connect new components to your board.
- Locate compatible XOD nodes for new devices.
- Outline some examples of how these skills can be applied to biological research.
- Recall where to find additional information and help with building your own devices.



# Hardware expansion

## Additional Components

The wide variety of low-cost components available for working with Arduino can be daunting, and understanding how to use and connect these components to your board can seem complex. Below we outline some of the simple systems available for connecting new hardware to your board, and on the next page we will discuss how to connect or wire up these components.

### GROVE COMPONENTS

The Grove board is made by open hardware company Seeed Studio, that provides a whole series of Grove components that are compatible with the Grove board via simple 'plug-and-play' connectors. These components can be plugged directly into the board using the white plug sockets in the middle of the board, and the cables provided in the kit. You can browse Grove-compatible components on the Seeed Studio website ([www.seeedstudio.com](http://www.seeedstudio.com) > Shop > Grove).

### M5STACK

M5Stack is another hardware company that sells its own Grove-compatible components which they term 'units'. You can browse M5Stack Grove-compatible components on the M5Stack website ([www.m5stack.com](http://www.m5stack.com) > Store > Unit).

### OPEN SMART COMPONENTS

The Biomaker Expansion kit is comprised of components from an alternative supplier, Open Smart. Open Smart provide both an 'easy-plug' system similar to the Grove system, as well as a wider variety of components which require simple, no-solder wiring. Grove plugs are not compatible with Open Smart plugs, so an expansion shield is required to connect these components to the board (see p77). You can browse Open Smart components on the Open Smart Ali Express store ([www.open-smart.aliexpress.com](http://www.open-smart.aliexpress.com)).

### OTHER COMPONENTS

Grove and Open Smart provide easy-to-use systems for connecting components to your Grove (or any other Arduino board), however, there are also a staggering variety of other suppliers and components available if you introduce a small amount of wiring and soldering. Companies such as Adafruit, SparkFun and Seeed Studio all provide useful electronics modules that can be easily connected to your board by soldering, or using a prototyping shield (see p76).

Seeed Studio (Grove), Adafruit and SparkFun are all based in the USA, and you can buy components directly from their websites, although customs fees and taxes are likely to apply. Fortunately, many of these components are also available from UK suppliers such as Farnell ([www.uk.farnell.com](http://www.uk.farnell.com)), Cool Components ([www.coolcomponents.co.uk](http://www.coolcomponents.co.uk)), RS Components ([www.uk.rs-online.com](http://www.uk.rs-online.com)) and Mouser Electronics ([www.mouser.co.uk](http://www.mouser.co.uk)).

M5Stack and Open Smart are based in China and you can buy components from their storefronts on Ali Express ([www.open-smart.aliexpress.com](http://www.open-smart.aliexpress.com) and [www.m5stack.aliexpress.com](http://www.m5stack.aliexpress.com)).



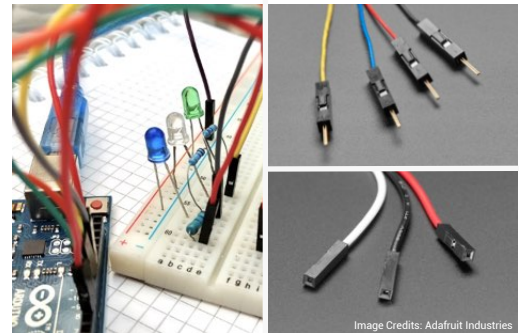
# Hardware expansion

## Connecting Components

There are a number of ways to connect additional components to your board, most of which make use of the board's 'header sockets'. These are the yellow plastic sockets arrayed around the left and right edges of the board's central module. Electronic components can be wired to these sockets (either directly or via a breadboard), added as part of a shield, or connected via a breakout board. Sometimes a combination of these methods is used. Below we explain each of these options.

### ELECTRONIC WIRING

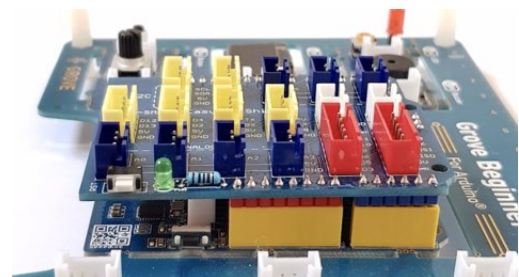
Each of the small header sockets on the board connects to one of the board's pins (see p11). You can wire components directly into these sockets, but more often a breadboard or shield is used. Breadboards are a way to easily make and test electronic circuits. You can learn more about them on the SparkFun website ([www.learn.sparkfun.com/tutorials/how-to-use-a-breadboard](http://www.learn.sparkfun.com/tutorials/how-to-use-a-breadboard)). The hook-up wires used to connect components usually come with two types of ends: male and female. Female ends are like individual sockets (similar to the header sockets), whilst male ends are metal pins, which fit into header and female sockets.



Left: Wiring to an Arduino using a breadboard  
Right: Male (top) and Female (bottom) wires

### SHIELDS

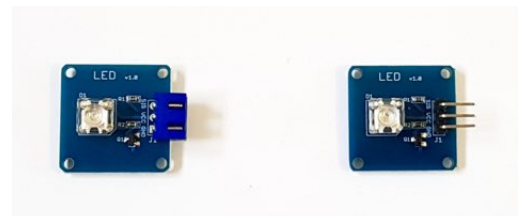
Shields are modular circuit boards that piggyback onto your Arduino to instill it with extra functionality. They use a series of header pins that fit directly into the header sockets. Shields can add a variety of functions, for example allowing the board to communicate via WiFi, adding extra storage capacity, or accessing GPS. Expansion and prototyping shields provide additional sockets or header pins that allow you to expand the capacity of your board and easily connect any number of custom components.



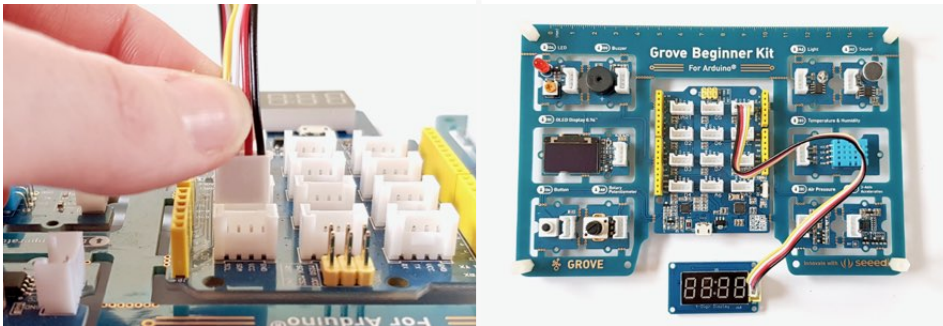
Open Smart shield connected to the Grove board

### BREAKOUT BOARDS

A breakout board is similar to a shield but usually with a single or small number of electronic components included. They are used to make wiring of components easier. Each of the modules (LED, buzzer etc.) on your Grove board is essentially a built in breakout board. It is often easiest to purchase components as part of a breakout board and then use electronic wiring or a shield to connect the breakout board to the Arduino.

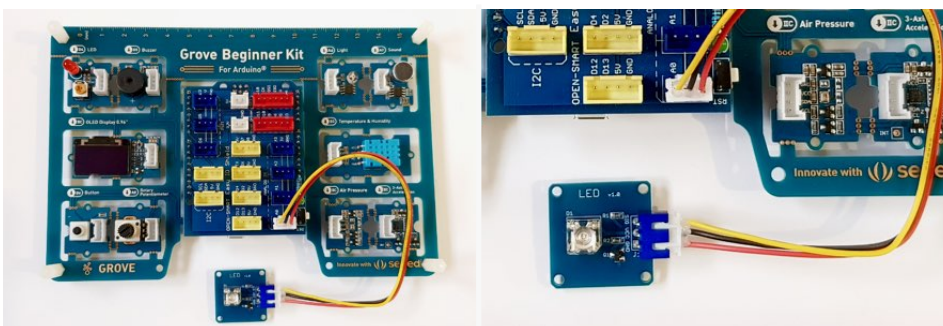


Open Smart easy-plug (left) and standard (right) LED breakout boards



### CONNECTING GROVE COMPONENTS

Grove components come as breakout boards with white Grove sockets included. You can use Grove cables (six are included in the Grove Beginner Kit) to connect them to the board. Simply plug one end into the breakout board, and one end into a white socket on the board. Note the name of the port you're using (A0, D5). This is written below the socket. Use A0, A2 and A6 for analog devices, D2-7 for digital devices and I2C for I2C devices.

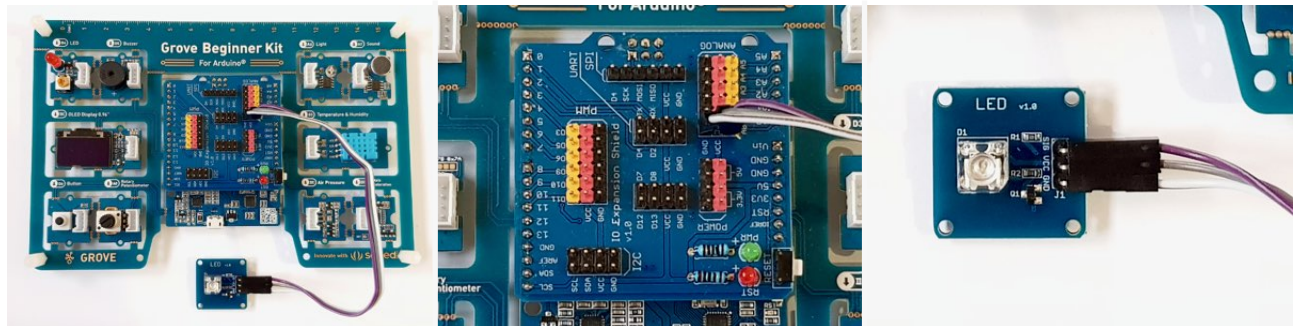


### CONNECTING OPEN SMART EASY-PLUG COMPONENTS

Open Smart 'easy-plug' components are similar to Grove components, but the plugs are not compatible. Adjust for this by plugging the easy-plug expansion shield into the header sockets, then use easy-plug cables to plug components in your breakout board. Again, note the name of the port you are using. Use A0-A3 for analog devices, D3, D5 and D6 for digital devices and I2C for I2C devices. For more information see the Biomaker website ([www.biomaker.org/s/Biomaker-Easy-Plug-Expansion-Kit-Information-Sheet.pdf](http://www.biomaker.org/s/Biomaker-Easy-Plug-Expansion-Kit-Information-Sheet.pdf)).



# Hardware expansion



## CONNECTING STANDARD OPEN SMART COMPONENTS

Standard Open Smart breakout boards come with male pins attached. You can use the Open Smart expansion shield and male-male wires to easily connect these components. Plug the Open Smart expansion shield into the header sockets, then use female-to-female wires to connect the pins on the breakout board to the pins on the expansion shield. Connect each pin to its corresponding pin on the expansion shield. E.g. connect VCC to VCC, GND to GND and SIG to a relevant pin (A0-5 for analog devices and D7,8,12 and 13 for digital devices). For I2C connect SDA to SDA and SCL to SCL. For more information see the Biomaker website ([www.biomaker.org/s/No-Code-Programming-for-Biology-Handbook.pdf](http://www.biomaker.org/s/No-Code-Programming-for-Biology-Handbook.pdf)).

## Connecting Other Types of Component

Breakout boards from other companies often come without connectors. You can add connectors by soldering header pins to them. Then you can use them like Open Smart components. For an excellent tutorial demonstrating this, see [www.rimstar.org/science\\_electronics\\_projects/pin\\_headers\\_soldering\\_cutting\\_male\\_female.htm](http://www.rimstar.org/science_electronics_projects/pin_headers_soldering_cutting_male_female.htm).

## Component conflicts

Because your Grove board already has components connected to many of the pins you may encounter clashes if you try to connect a second device to the same pin. This may or many not disrupt your programme, depending on the devices involved. To avoid this, prioritise use of pins without devices already attached, such as D2. If clashes become an issue you can use a different Arduino, such as the Arduino Rich Uno R3 (provided in the Biomaker expansion kit), or you could detach the central module of the Grove board to use separately. Note that clashes are not an issue with I2C devices as they can be connected to the same pins and identified via their addresses.





# Documenting circuit construction

## Graphical description of circuits

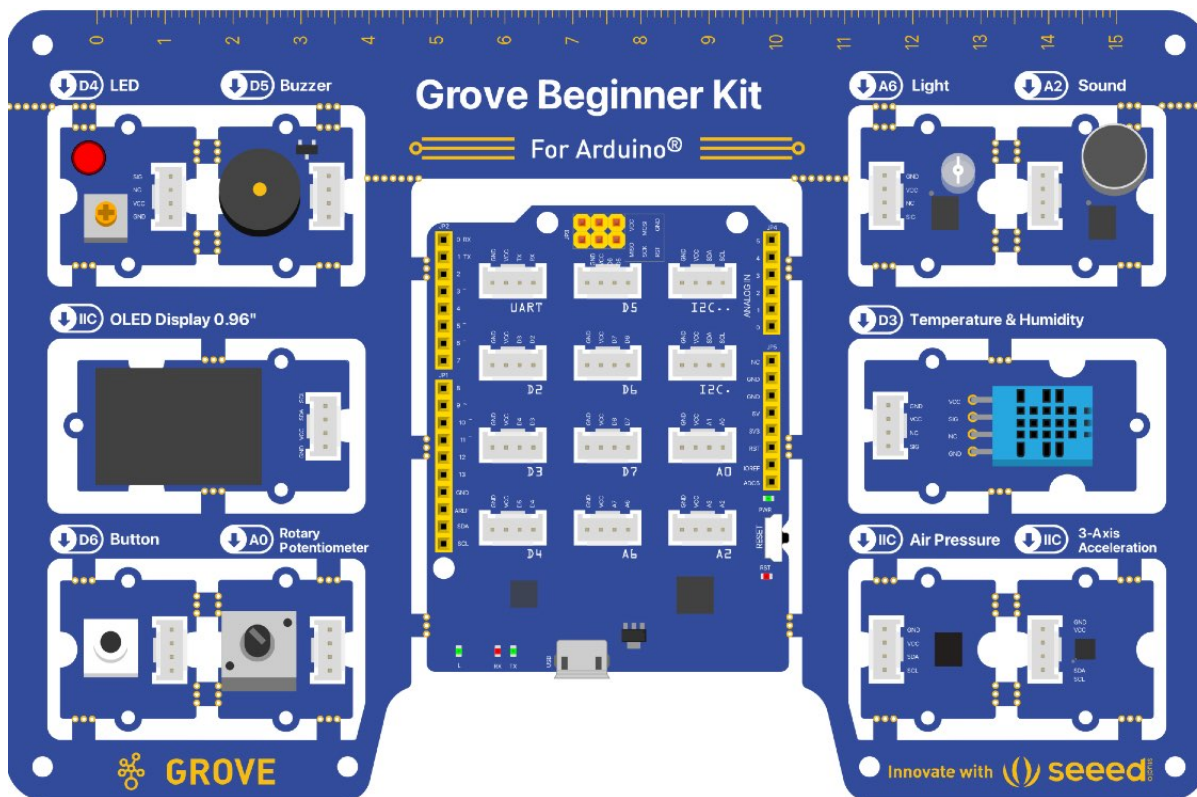
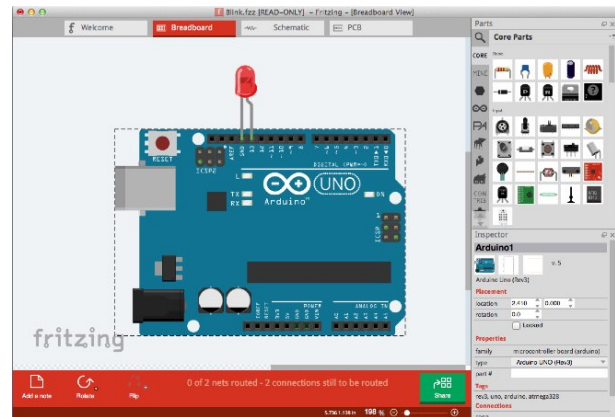
Fritzing is an open-source software tool that allows users to design, document, and share electronic circuits. It is particularly popular among hobbyists, makers, and educators due to its user-friendly interface and ability to generate circuit schematics and breadboard layouts.

The breadboard view allows users to create virtual circuits using a breadboard-like interface. Users can drop-and-drag components, such as Arduino boards, sensors, and other electronic parts, and connect them using virtual jumper wires. This view is particularly helpful for planning and documenting prototype circuits before physically building them on an actual breadboard.

Fritzing has a built-in library of commonly used electronic components, and users can create and import custom components if needed. The software also allows users to share their designs with the Fritzing community, making it a good resource for learning and collaboration.

Biomaker commissioned the construction of a Fritzing part for the Grove Beginner board, which can be downloaded at <https://www.biomaker.org>: (Grove\_Beginner\_Kit.fzpz). An example is shown below: before and after the wiring of an external I2C-connected component (1602 LCD display) to the Seedstudio Grove Beginner board.

Fritzing can be downloaded at: <https://fritzing.org>

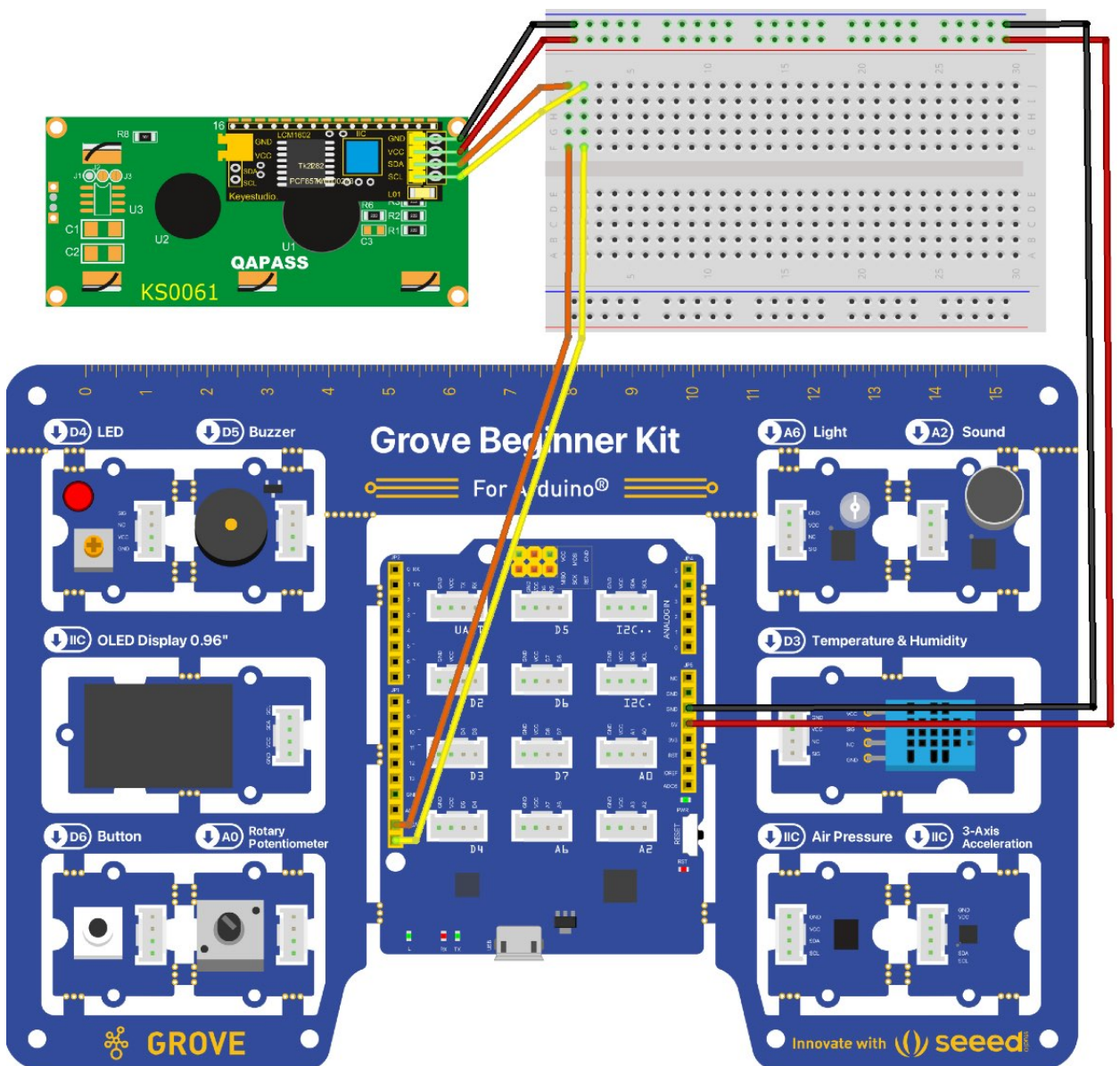


fritzing

## Biomaker project archive on Hackster.io

We have adopted Hackster.io as the web platform for documenting Biomaker projects. Hackster provides a simple menu-driven interface for easily assembling project descriptions. As projects are entered, the identities of parts are entered, and these provide a link to other projects that use the same components. The individual or team that owns a project are credited with this and each participant can automatically build a portfolio of projects that they have participated in - potentially a very useful addition to a CV.

Log in and create a project using any preliminary work as a guide (select the blue "submit a project" button). Project descriptions are private at first, and can be expanded over the course of the work. Projects can be branched to create a portfolio of related activities, and ultimately published. This has the effect of advertising your project to the over 2 million users of Hackster, with a 1-page project description that can be made accessible freely across the Internet. You can submit your project to the Biomaker platform (<https://www.hackster.io/biomaker>). Hackster handles the mechanics of integrating your project with 10,000's of existing projects. Hackster provides a knowledge base with a wide range of helpful information to help get started (<http://help.hackster.io/knowledgebase>).

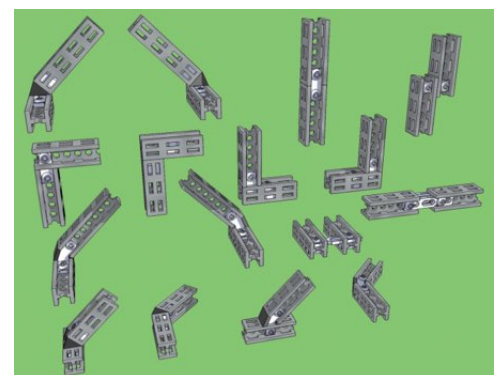
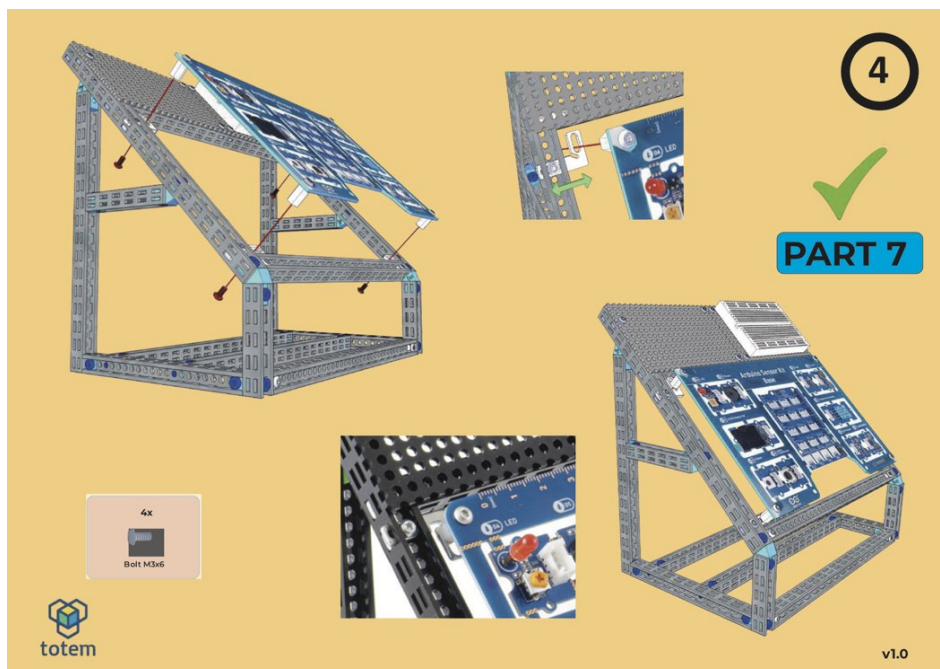
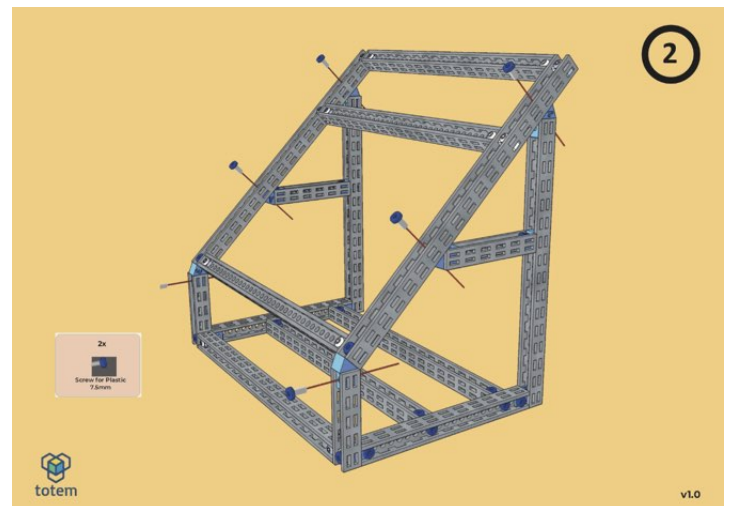
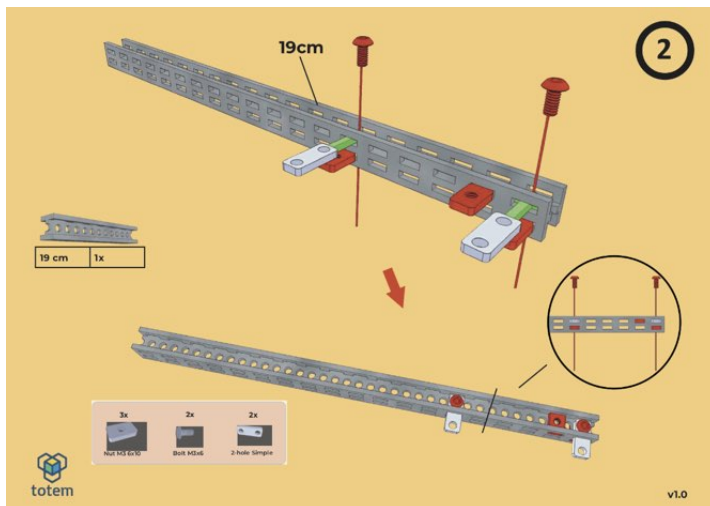


# Hardware stands

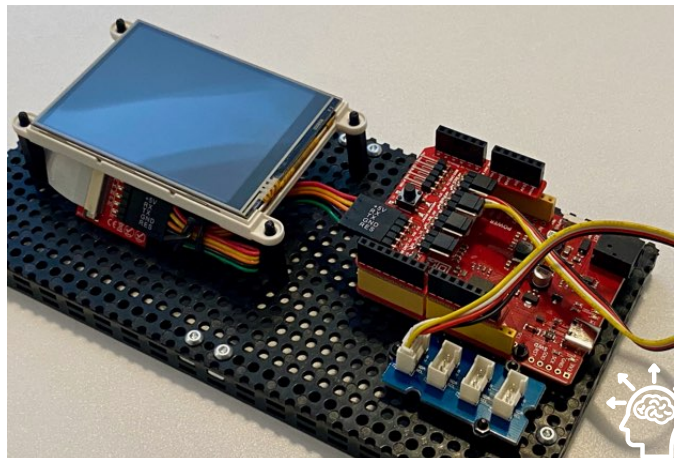
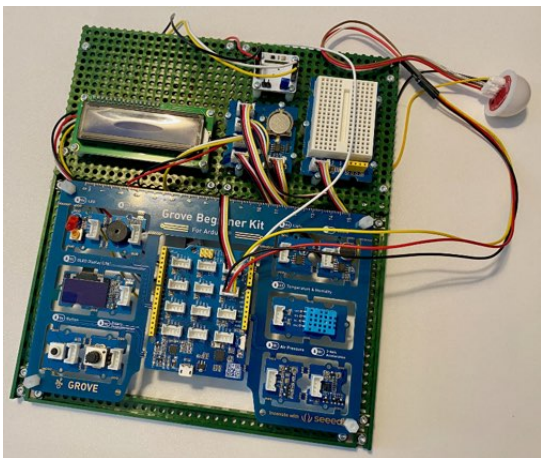
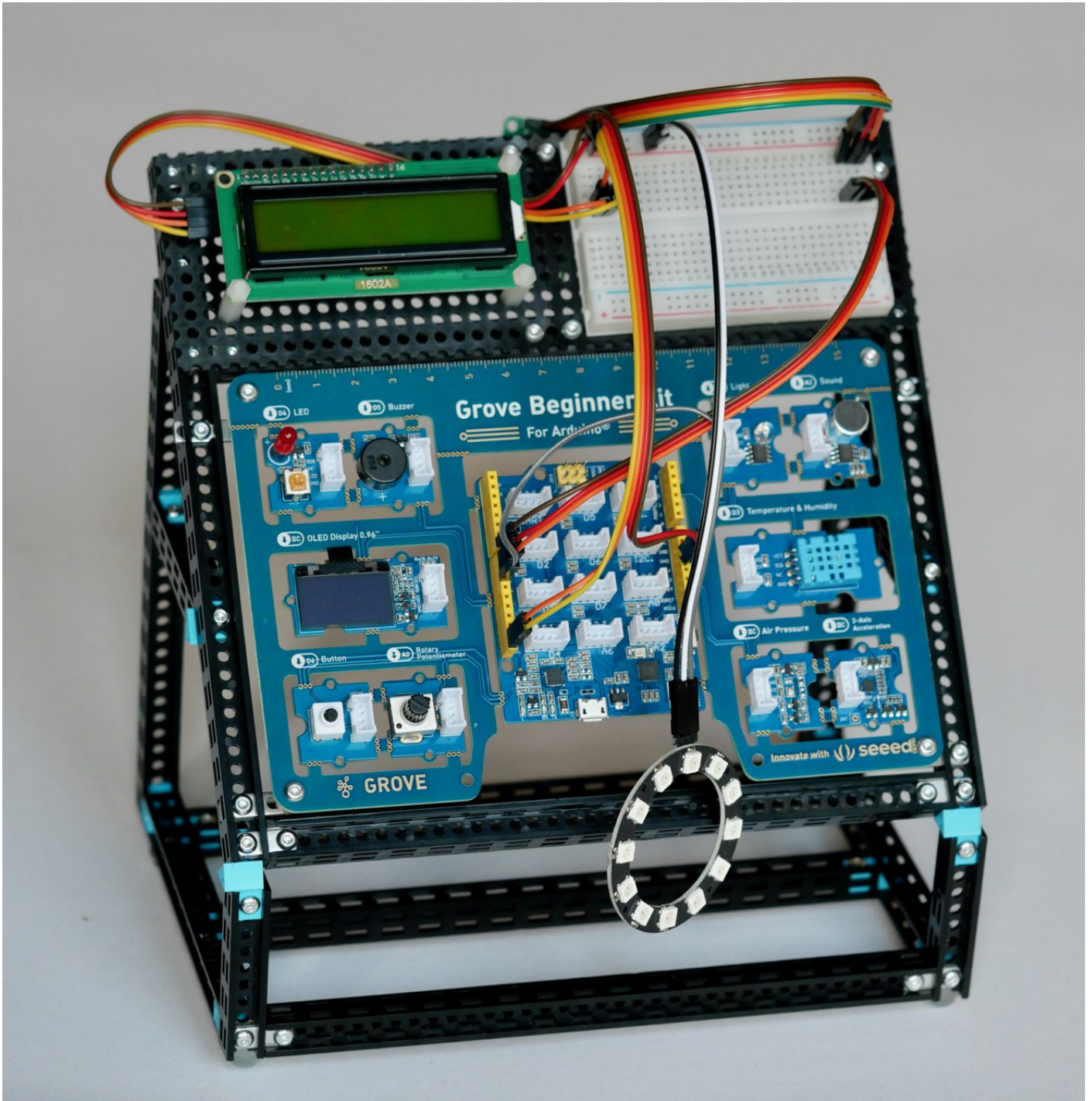
Totem are an engineering company based in Lithuania (<https://totemaker.net>) who have created a unique system for highly-adaptable construction of small scale elements like instrumentation chassis and robots. The design of the Totem system is based on several principles: (i) key parts are plastic beams that can be easily cut to custom dimensions, (ii) the beams are connected via modular metal connectors to create complex and sturdy structures, (iii) the building blocks have dimensions in 5 and 10 mm increments for simple assembly, (iv) brackets allow simple integration of third-party parts, like electronics, mechanics and custom 3D printed parts. The system is very well suited for DIY projects and rapid construction of customised prototypes, including prototype instrumentation. Totem distributes a wide range of part and project kits worldwide.

The engineers at Totem have put together a low-cost kit specifically to hold the Seeed Studio Grove Beginner Kit. Details can be found at: (<https://totemaker.net/product/totem-rack-for-grove-beginner-kit>). This stand is useful for mounting the Beginner Kit board along with a mini breadboard and space for mounting other components. The stand is useful in an educational context providing a more rigid package, and keeping loose parts together in a shared environment. Another benefit is that the customisable nature of the Totem components allows easy accommodation of other components and form factors (see facing page). Complete TotemMaker Kits are available for flexible designs (<https://totemaker.net/product/totem-maker-kit>).

Below: Images from the Totem instruction manual for self-assembly of the Grove Beginners Kit rack. Step-by-step instructions are provided. Inset shows some of the various joints that can be constructed from cut plastic beams and modular metal connectors.







# Connecting external hardware

There are several electrical standards for connecting electronic devices to an Arduino board, depending on the specific requirements of your project. Here are some common methods:

**Digital Input/Output pins:** These pins can be used to connect devices that require simple on/off signals, such as LEDs, switches, or buttons. You can set these pins as INPUT or OUTPUT using the `pinMode()` function in the Arduino code.

**Analog Input pins:** Analog input pins allow you to connect devices that produce a range of voltage levels, like sensors (e.g., temperature, light, or sound sensors). The Arduino can read these varying voltage levels using the `analogRead()` function and process the data accordingly.

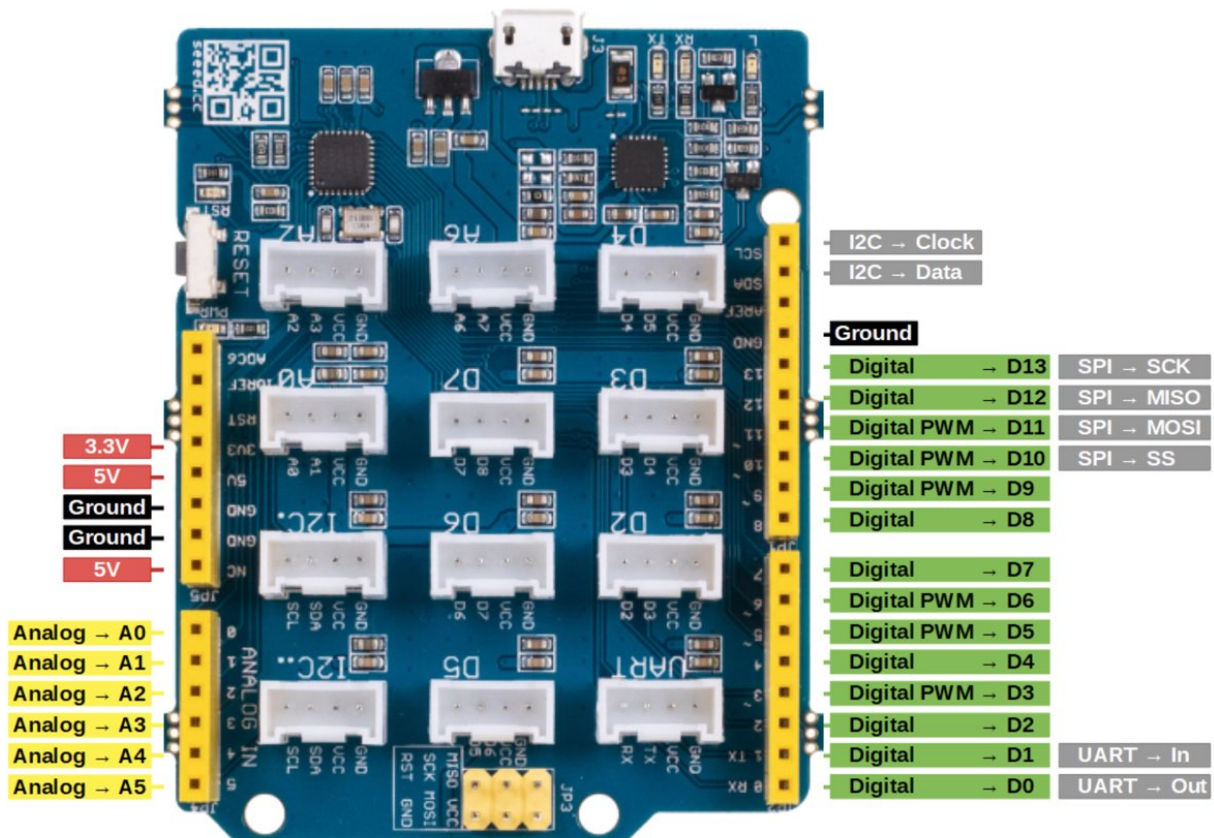
**PWM (Pulse Width Modulation) pins:** Some digital pins on the Arduino can produce PWM signals, which can be used to control devices like servos, motors, or LED brightness. You can generate PWM signals using the `analogWrite()` function in your code.

**Serial Communication:** The Arduino board supports serial communication through its TX (transmit) and RX (receive) pins. This enables communication with devices like GPS modules, Bluetooth modules, or other microcontrollers. You can use the Serial library in your Arduino code for serial communication.

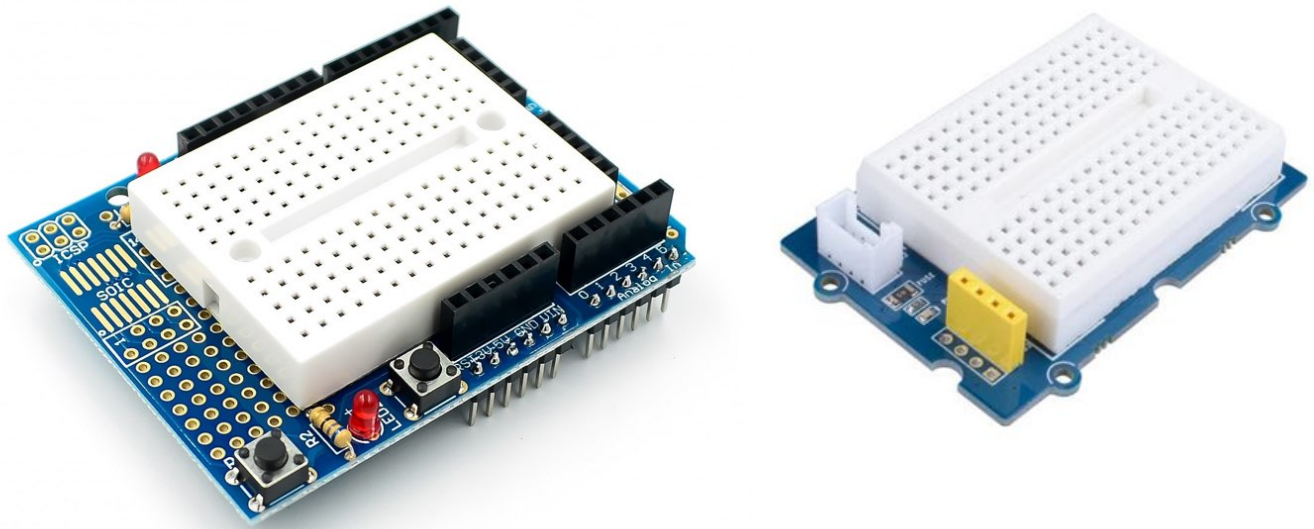
**I2C (Inter-Integrated Circuit) Communication:** I2C is a communication protocol that allows multiple devices to be connected using only two pins (SDA for data and SCL for the clock). This is useful when connecting devices like displays, sensors, or EEPROM memory chips. The Wire library is used to implement I2C communication in Arduino code.

**SPI (Serial Peripheral Interface) Communication:** SPI is another communication protocol that uses a master-slave configuration. It requires four pins: MISO (Master In Slave Out), MOSI (Master Out Slave In), SCK (Serial Clock), and SS (Slave Select). SPI is often used with SD cards, displays, and sensors. The SPI library can be used to implement SPI communication in your Arduino code.

Choose the appropriate method based on your project's requirements and the specific electronic devices you want to connect to your Arduino board. It's also crucial to consider the power requirements of the devices, ensuring the correct voltage supply (3V vs 5V) and that power consumption doesn't exceed the Arduino's power supply limits.





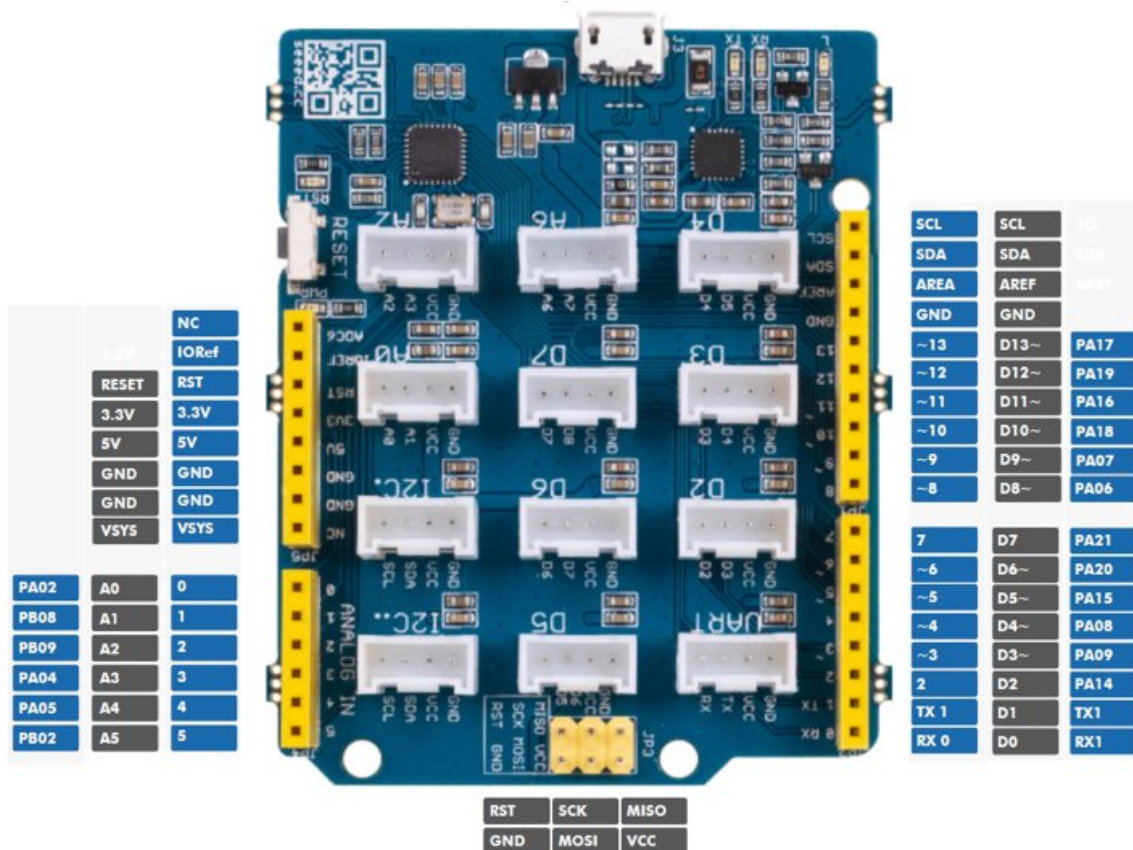


A **breadboard** is a useful tool for physically connecting components to an Arduino board without soldering. It allows you to create temporary circuits and test your designs before finalising them.

Place components on the breadboard, inserting the leads of components into the breadboard's holes. Each row of holes is electrically connected, forming a node. When placing components, ensure their leads do not share the same node to avoid short circuits. Some breadboards have power rails that run along the sides of the breadboard and provide 3V or 5V power and ground connections for your components.

Two simple solutions are shown above, with mini breadboards mounted on an Arduino-compatible shield (left) and a Seed Studio Grove board (right). Both allow direct connection to the Beginner board, and assembly of small custom circuits.

Breadboards are intended for prototyping and testing circuits. Once you've verified that your circuit works correctly, you can create a more permanent solution by soldering components on a modular copper-surfaced perfboard or by designing a custom printed circuit board (PCB).



# Electrical connections

## Grove connectors

The microcontroller board in the Beginner Kit is covered with 12 Grove connectors (see facing page). The Grove connector standard was developed by Seeed Studio as a modular, easy-to-use system for connecting electronic components. It simplifies prototyping and development by eliminating the need for soldering and providing a complement to breadboarding.

The Grove system uses 4-pin connectors with a standardised 2.0mm pitch, which carry power (VCC and GND) and either digital or analog signals, depending on the module. There are 4 different types of Grove connector interface that provide access to digital, analog, I2C, and UART ports, respectively.

The Grove connector has four pins numbered one through four, as shown in the image opposite. The cables are color-coded: red is power and black is ground; this is always the case no matter what peripheral you connect, but pins 1 (yellow) and 2 (white) vary in function according to the Grove module being connected.

- **pin 1 - Yellow** (Primary digital I/O or analog input, SCL on I2C connector, or Rx serial receive on serial port)
- **pin 2 - White** (Secondary digital I/O or analog input, SDA on I2C Grove connector or Tx serial transmit on serial port))
- **pin 3 - Red** - VCC on all Grove connectors (5V for the Beginner Kit board, but can be 3V on other microcontroller boards)
- **pin 4 - Black** - GND on all Grove connectors

The Grove connector cables are identical, just carrying different signals. Cables can only be inserted into the corresponding plugs in a one way, so no chance of mis-wiring.

Seeed Studio manufacture a wide range of Grove modules, including sensors, actuators, displays, and communication devices. Generally, these can be directly plugged into a suitable corresponding Grove socket and is then software-accessible from the microcontroller. However, it is often convenient to also access other devices that are not Grove compatible (see below). These might be newer or cheaper devices accessible via pins on a breakout board. Aliexpress is a convenient source of cheap electronic devices in this format. In these cases, the Grove Breadboard - a mini breadboard on a printed circuit board with in-built Grove connector (see previous page) can be very useful.

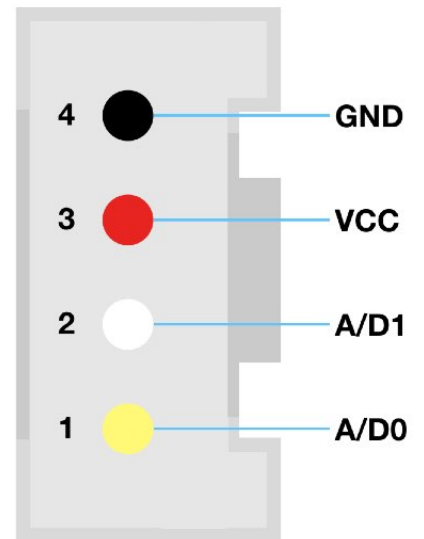
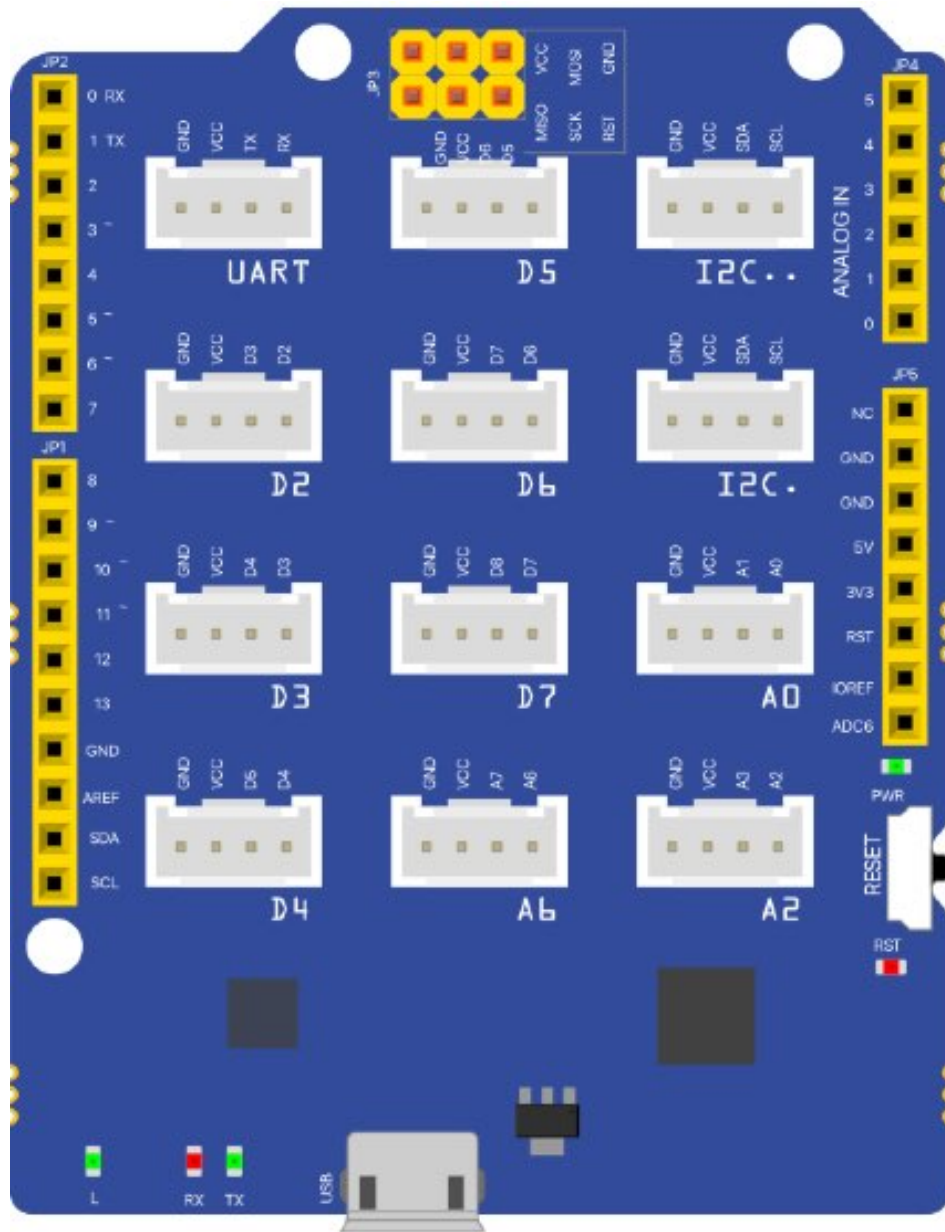
Alternatively, devices can be connected via Grove cables that have open leads or sockets on the component side, or the devices can be connected via leads to the Arduino Shield sockets at the sides of the microcontroller board.

### Digital port(s)

pin	Function	Note
pin1	Dn	Primary Digital Input/Output
pin2	Dn+1	Secondary Digital Input/Output
pin3	VCC	Power for Grove Module, 5V/3.3V
pin4	GND	Ground

### Analog port(s)

pin	Function	Note
pin1	An	Primary Analog Input
pin2	An+1	Secondary Analog Input
pin3	VCC	Power for Grove Module, 5V/3.3V
pin4	GND	Ground



## I2C port

pin	Function	Note
pin1	SCL	I2C Clock
pin2	SDA	I2C Data
pin3	VCC	Power for Grove Module, 5V/3.3V
pin4	GND	Ground

## Serial port

pin	Function	Note
pin1	RX	Serial Receive
pin2	TX	Serial Transmit
pin3	VCC	Power for Grove Module, 5V/3.3V
pin4	GND	Ground



# 16x2 character LCD screen

## Description

A 1602 liquid crystal display (LCD) is capable of displaying 2 lines of 16 characters, and is a very useful device for user feedback in instrumentation design and debugging. The device can be obtained with an I2C interface backpack (recommended over versions that retain a multi-connector parallel interface) that allows serial communication with the device. (This is the black-coloured circuit board soldered to the back of the green-coloured LCD board). The I2C interface allows communication with the LCD screen through two wires plus power supply, rather than 8+ wires required by a parallel port device. The I2C protocol allows communication with multiple devices on the same 2 wire bus. Each device needs a unique address, usually set in the hardware.

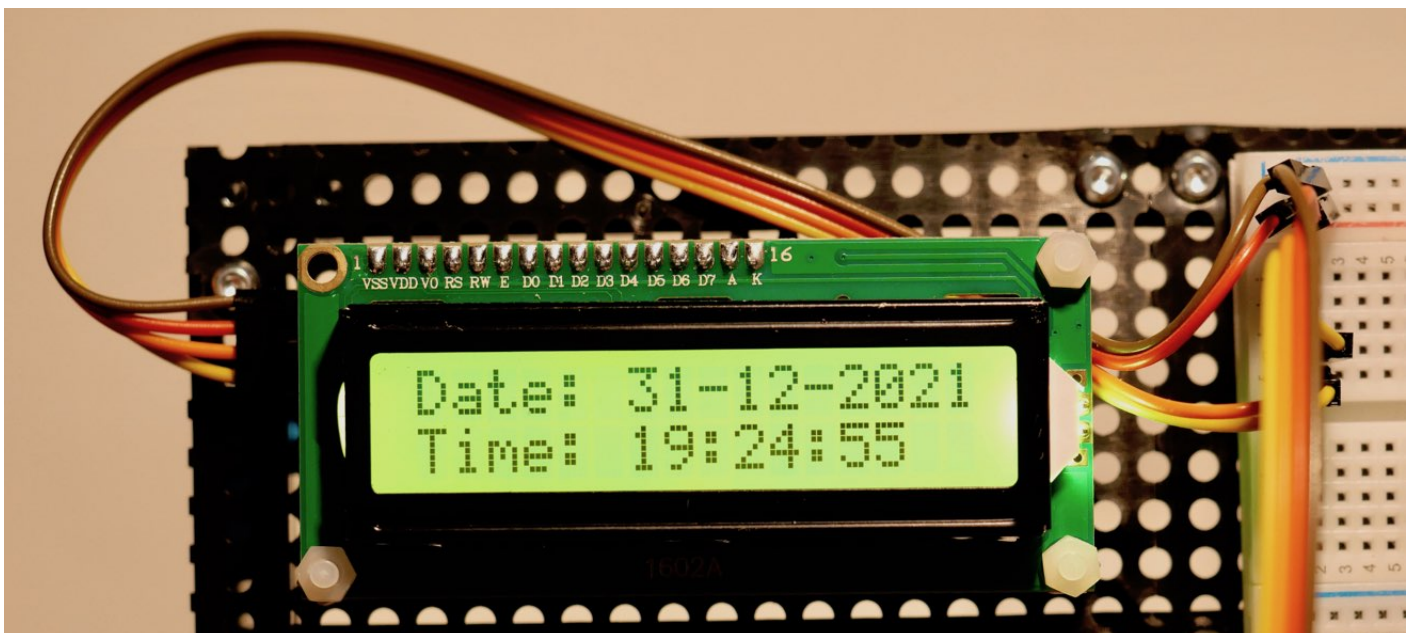
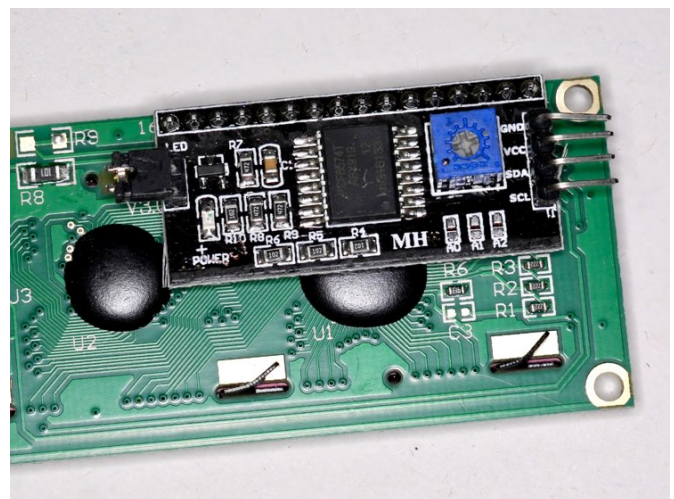
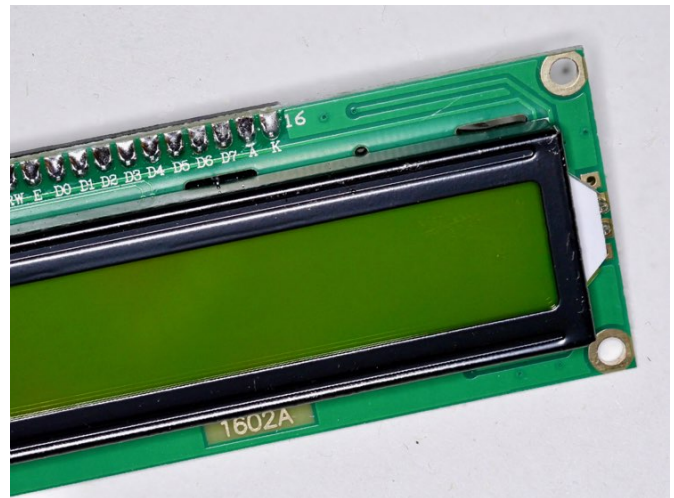
Compatible devices are widely available from component suppliers. For example, Biomaker sourced the I2C LCD1602 Green from the TZT store on Aliexpress (<https://www.aliexpress.com/item/1005001609811389.html>) for Stage 2 expansion kits,

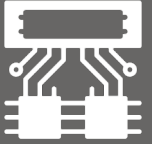
## Specifications

The LCD display is powered by a 5V supply and draws about 25mA with the backlight on, and 2mA without. The green coloured backlight sits behind black coloured characters. The characters are formed in two lines of 16 characters in 5x7 dot matrices.

**Advanced use:** If you wish to use multiple LCD displays on the I2C bus, you can add solder bridges to the jumpers A0, A1 and A2 on the I2C backpack - in order to change the address of each device, and allow them to be individually addressed. The supplied I2C backpack has a PCF8574T chip: and the IC address is (high order first) 0100 A2 A1 A0. When shipped, A2~A0 are all vacant. The default I2C address therefore: 0100 111 (0x27). If you want to modify the address yourself add the relevant jumpers, noting that floating address pad is 1, and the short circuit is 0 after adding a solder bridge.

**Important:** There is a potentiometer that controls the contrast setting of





the display. It is controlled by a black plastic wheel at the front left edge of the LCD screen. (Contrast can also be adjusted using the blue potentiometer on the I2C backpack). The contrast setting requires fine adjustment, and the screen will appear blank and unresponsive if badly adjusted. If, on first use, you want to check that the LCD screen is correctly connected (i.e. are using the correct I2C address), use a XOD node to switch the backlight on and off. If that works, load some text into the screen, and adjust the contrast for best legibility.

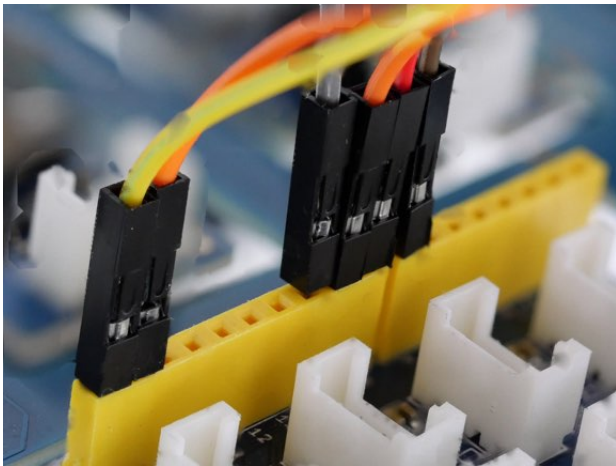
## How to Connect

The display should be connected to the microcontroller via the Vcc (5V), GND (ground), SDA (data) and SCL (clock) wires. Because it is common to use multiple I2C devices, it is generally easier to connect through the breadboard, which allows multiple devices to share the I2C signals and power from the relevant sockets on the yellow connector on the microcontroller board.

To connect to the Grove Beginner Kit board:

Use Dupont cables or 22AWG solid core hookup wire to connect the SDA, SCL, 5V and GND signals on the main yellow connector for the Arduino-compatible microcontroller to a mini breadboard. Use connected sockets on the breadboard to further connect to the 4 pins on the I2C backpack of the LCD display.

(Optional) Attach the LCD display module to the plastic mounting sheet on the Totemmaker stand. For example, 3x M3 bolts and stand-offs will work well - or you can use stick-down Velcro or other temporary measures.



## XOD support

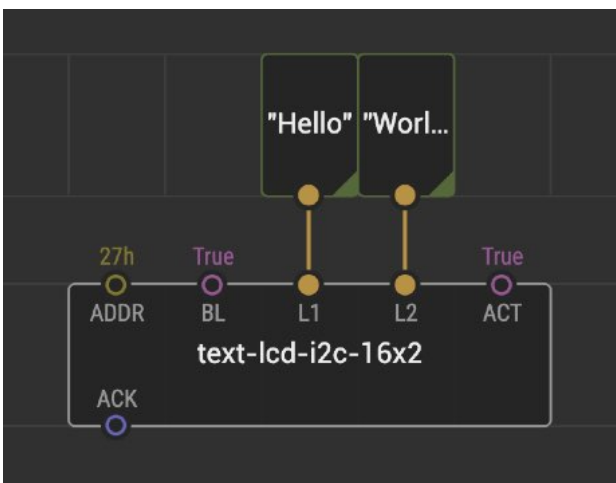
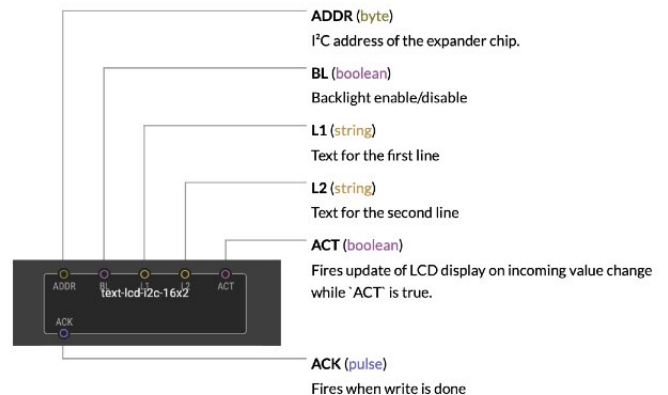
XOD provides the software node `text-lcd-16x2-i2c`, that allows direct communication with the display, with inputs for each line of the display (see below). The address of the I2C device should be set at 27h using the `ADDR` parameter.

The text display can be a very useful tool for following program behaviour. In addition, XOD provides the `watch` node, a number of which can be connected to the outputs of key nodes, and provide real-time output of values as a programme is run in debug mode.

### text-lcd-i2c-16x2

[xod-dev/text-lcd/text-lcd-i2c-16x2](#)

A quick-start node drives a text LCD screen with a PCF8574 or PCA8574 I<sup>2</sup>C expander module. Usually have a value in range 0x20-0x27 or 0x38-0x3F. Consult LCD/expander documentation to know the exact value.





# Using a 16x2 character LCD screen

## XOD library

XOD provides a standard library (**xod-dev/text-lcd**) to drive the screen. This includes useful nodes like **text-lcd-i2c-16x2** for directly displaying text, as well as other specialised nodes and examples (right).

The **xod-dev/text-lcd** library provides support for interfacing with different text LCD displays, both 16x2 and 20x4 character screens, which are commonly used in Arduino projects. Here are some key features available in the library:

1. **text-lcd-i2c-16x2** and **text-lcd-i2c-20x4**: These nodes are used to represent the 16x2 and 20x4 character I2C-connected LCD displays, respectively. They provide a simple way to configure and interface with these types of displays.
2. **Print-at**: This node allows you to display text on the LCD screen at a defined row and position.
3. **clear**: This node clears the entire display, removing any text or symbols previously shown.
4. **backlight**: This node controls the backlight of the LCD display. You can set the brightness level by providing a value between 0.0 (off) and 1.0 (full brightness) to the LUM input.

For example: scroll down the XOD user interface to find the **xod-dev/text-lcd** library, open and identify the **text-lcd-i2c-16x2** node and drag onto the work area. The node allows setting parameters for the screen, and feeding data to the display.

(The node provides the code necessary to drive a text LCD screen with a PCF8574 or PCA8574 I2C expander module. These usually have a value in the range 0x20-0x27 or 0x38-0x3F. Consult LCD/expander documentation if you want to use a different LCD display and need to know the exact value. Also, Cesar Sosa has provided a simple XOD patch that can be used to scan the I2C bus for connected devices: <https://forum.xod.io/t/scanner-device-i2c/1490>, and can be loaded as a XOD library **cesars/i2c-scanner**)

## Further information:

Working with text displays: <https://xod.io/docs/guide/text-lcd/>  
 XOD tutorial: <https://xod.io/docs/tutorial/108-text-lcd/>  
 I2C communication basics: <https://xod.io/docs/guide/i2c/>  
 Details of jumper selections for adjusting display address, etc: <https://lastminuteengineers.com/i2c-lcd-arduino-tutorial/>

Specifications for I2C backpack for 1602 display: [https://handsontec.com/dataspecs/module/I2C\\_1602\\_LCD.pdf](https://handsontec.com/dataspecs/module/I2C_1602_LCD.pdf)

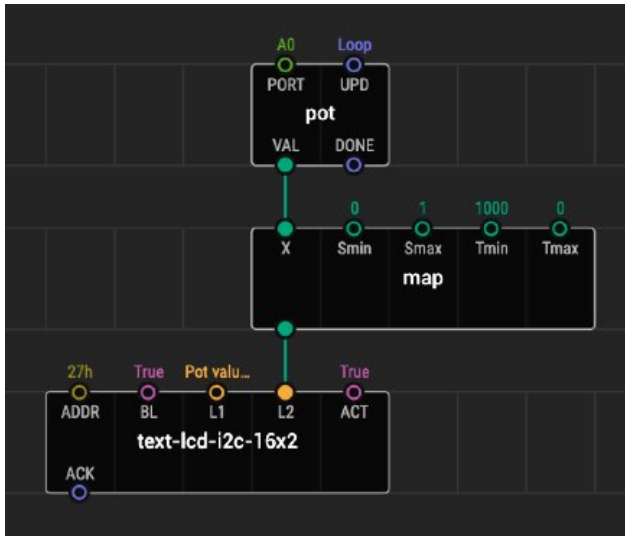
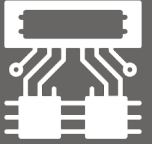
## xod-dev/text-lcd@0.37.3

License: AGPL-3.0

Nodes to drive a common text liquid crystal displays with I<sup>2</sup>C or parallel interfaces.

Node	Description
<a href="#">clear</a>	Clears the LCD screen and positions the cursor in the upper-left corner
<a href="#">clear(text-lcd-i2c-device)</a>	Clears the LCD screen and positions the cursor in the upper-left corner
<a href="#">clear(text-lcd-parallel-device)</a>	Clears the LCD screen and positions the cursor in the upper-left corner
<a href="#">example-i2c-print</a>	No description
<a href="#">example-parallel-print</a>	No description
<a href="#">example-print-at</a>	No description
<a href="#">example-quick-start</a>	No description
<a href="#">print-at</a>	Prints a text on the LCD screen in the allocated area, specified by 'ROW' index, position at the row and length. The text trims to the allocated area length. If the text is shorter – the rest of the allocated area will be cleared (replaced with whitespaces).
<a href="#">print-at(text-lcd-i2c-device)</a>	No description
<a href="#">print-at(text-lcd-parallel-device)</a>	No description
<a href="#">set-backlight</a>	Turns on or off the backlight of the I <sup>2</sup> C text LCD.
<a href="#">text-lcd-i2c-device</a>	No description
<a href="#">text-lcd-i2c-20x4</a>	A quick-start node drives a text LCD screen with a PCF8574 or PCA8574 I <sup>2</sup> C expander module. Usually have a value in range 0x20-0x27 or 0x38-0x3F. Consult LCD/expander documentation to know the exact value.
<a href="#">text-lcd-parallel-16x2</a>	Drives a text LCD screen with HD44780 chip.
<a href="#">text-lcd-parallel-20x4</a>	Drives a text LCD screen with HD44780 chip.
<a href="#">text-lcd-parallel-device</a>	No description
<a href="#">text-lcd-i2c-16x2</a>	A quick-start node drives a text LCD screen with a PCF8574 or PCA8574 I <sup>2</sup> C expander module. Usually have a value in range 0x20-0x27 or 0x38-0x3F. Consult LCD/expander documentation to know the exact value.

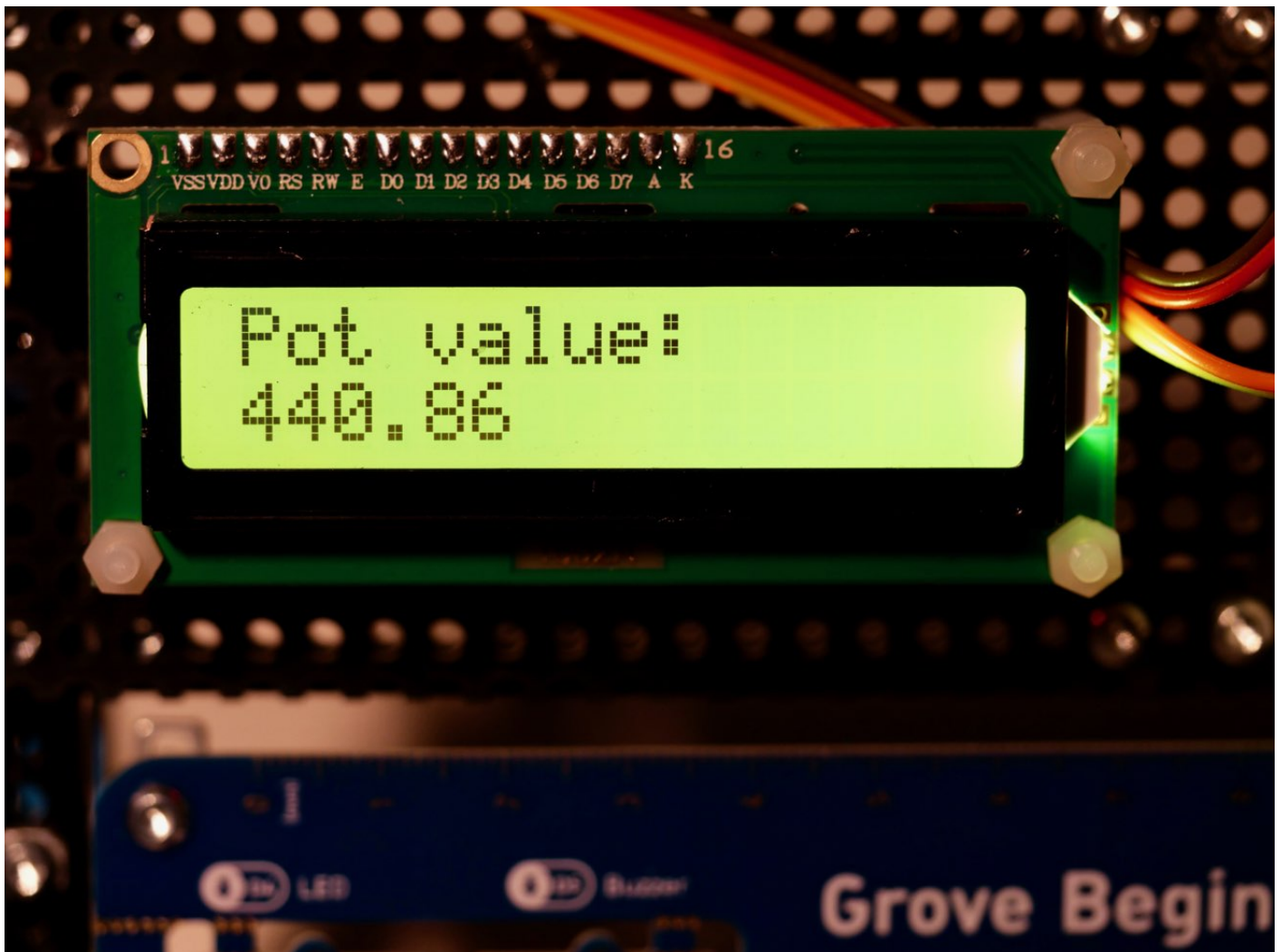




## XOD patch to test the device

A simple way of testing the connected display is to add **xod/core/constant-string** nodes, and use these to feed text to the L1 and/or L2 input ports of the **text-lcd-i2c-16x2** node.

Alternatively, **xod/common-hardware/pot** node can be used to generate varying values (between 0-1), and these fed to a **xod/math/map** node to recast these between arbitrary values. The patch shown left produces values between 0 and 1000 depending on the position of the potentiometer. These can then be displayed using the **text-lcd-i2c-16x2** node, and in this case, shown with a fixed text label in the first line of the display (L1).



# 12 x WS2812 RGB LED Ring

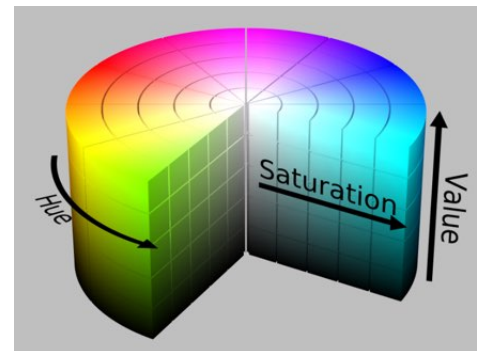
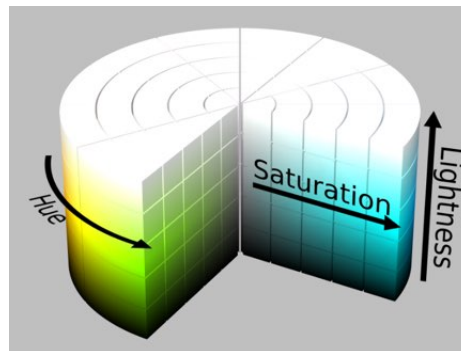
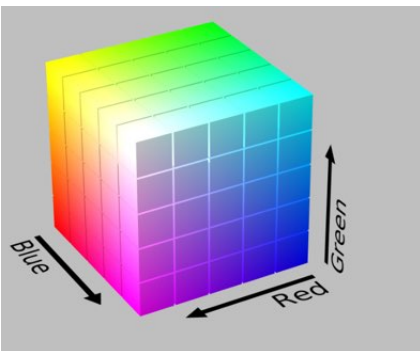
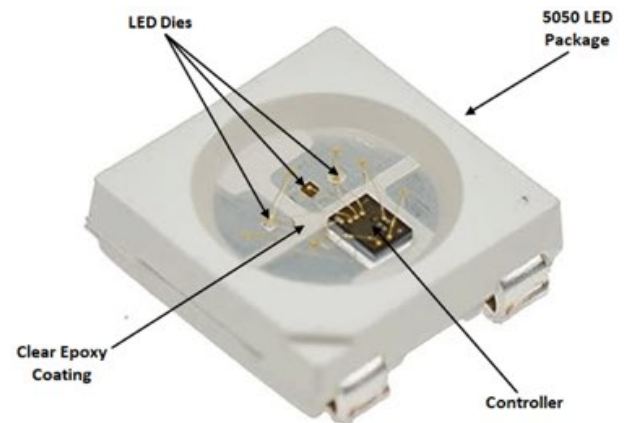
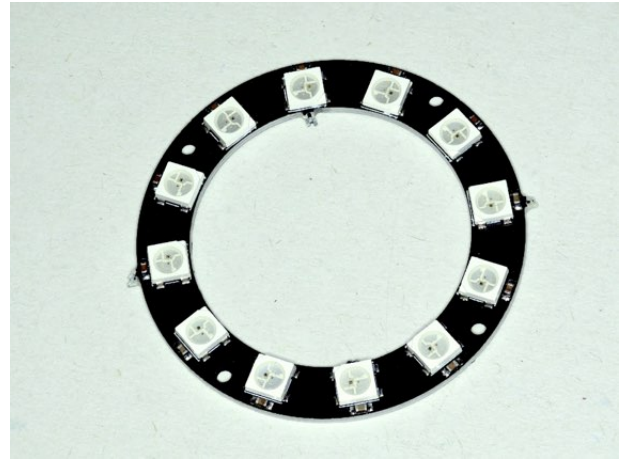
## Description

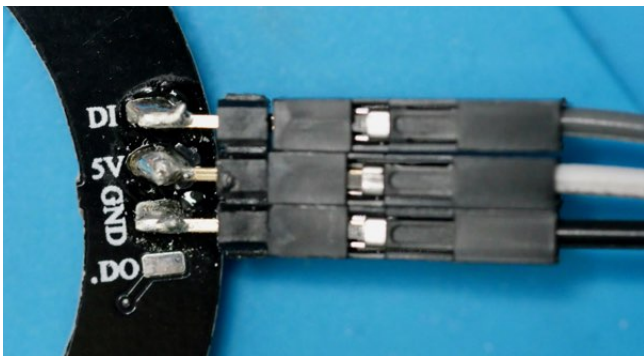
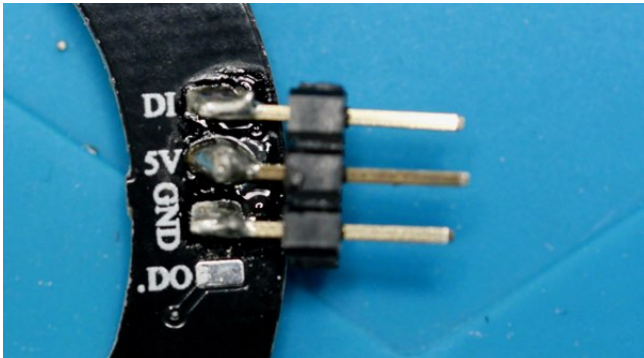
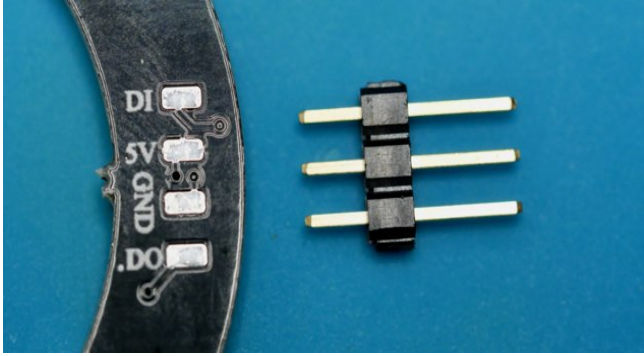
The LED Ring contains 12 addressable WS2812B RGB LEDs (Red-Green-Blue Light Emitting Diodes), which can each be programmed from the microcontroller. The LED ring provide illumination required in machine vision, biomedical, fluorescence, and strobing applications - or be used as a vivid indicator or alert for sensors and logic programs. The use of 12 LEDs allows it to closely mimic a clock face.

The WS2812 is a popular and widely used RGB LED that integrates an LED chip and a control circuit into a single package. The WS2812 is often referred to as a "NeoPixel," which is Adafruit's brand name for individually addressable RGB LEDs based on the WS2812 and similar chips. These LEDs enable you to create colorful lighting effects and animations by controlling each LED's color and brightness individually.

Here's an overview of how the WS2812 RGB LED works:

1. **Integrated LED and control circuit:** The WS2812 features an integrated design, combining an RGB LED and a control circuit in a single 5050 (5mm x 5mm) surface-mount package. This compact design simplifies the manufacturing process and reduces the number of external components needed to drive the LED.
2. **Single-wire communication:** The WS2812 uses a single-wire communication protocol, requiring only one data pin to control multiple LEDs. You can chain multiple WS2812 LEDs together by connecting the data output (DO) pin of one LED to the data input (DI) pin of the next LED. This allows you to create long LED strips or complex arrangements with a large number of LEDs while minimizing the number of pins required for control.
3. **24-bit color control:** Each WS2812 LED can display 16.7 million colors, thanks to its 24-bit color control (8 bits per color channel: red, green, and blue). You can set the color and brightness of each LED individually by sending a 24-bit data packet containing the desired color values.
4. **Timing-based protocol:** The WS2812 communication protocol is timing-based, relying on precise pulse widths to transmit data. A "0" bit is represented by a short pulse (approximately 400 ns high), and a "1" bit is represented by a longer pulse (approximately 800 ns high). The data is sent in a specific order: green, then red, and finally blue. The first LED in the chain receives the data, processes it, and passes the remaining data to the next LED. This continues down the chain until all LEDs have received their respective data.





## How to Connect

It only requires 3 connections for the 12 LEDs (VCC (5V), GND (ground) and DI - data 'in' pin). Data for LEDs is sent in serial, 24 bit for each LED, representing 8 bit for the red, green and blue LED housed in each device. The controller contains a 24-bit register, which takes in serial data from the DIN pin and stores and decodes and displays it on the respective LED.

The 24-bit register is divided into three parts, each one is 8 bits long and holds a different brightness value for each color. Since there are 8 bits, there can be 256 possible brightness values for each LED. As there are three colors, a total of close to 17 million colors are possible.

The data pins on the LED are designed to be daisy-chained; the output of each device's controller is buffered to maintain signal quality even if many LEDs are connected.

Wiring may require access to a soldering iron. The RGB LEDs are connected in serial fashion, and each has a particular position in the linear sequence. The microcontroller can communicate with each and all LEDs in sequence. In addition, LED rings (or other LED sticks or arrays) can be daisy-chained to build interconnected LED sequences.

Each LED is individually addressable via serial digital communication that is passed into the ring via the DI (Digital In) pin, and can be passed through to another device via the DO (Digital Out) pin. If we are using the ring as a solitary device, we don't need to use the DO pin.

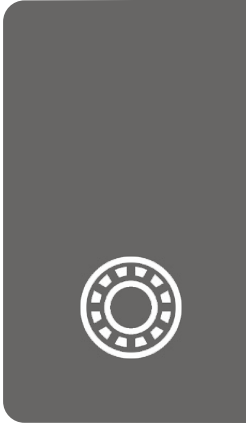
In order to connect the LED ring to the microcontroller, we need to connect leads to the DI pad and supply power through the 5V and GND (ground) pads. This can be done by directly soldering wire leads to the relevant pads, or soldering pins, that allow connection via socketed Dupont leads (shown left).

To connect to the Grove board:

1. Select an unused digital port on the microcontroller and connect this to the DI port of the LED ring, either directly or via the breadboard.
2. Connect the ground (GND) pin to a ground socket on the microcontroller board, either directly or via the breadboard.
3. Connect the 5V (Vcc) pin to a 5V source on the microcontroller board, either directly or via the breadboard.

Note: the LED ring is powered via the 5V supply from the microcontroller board, which is in turn usually powered through a USB port on a laptop or equivalent. If you notice erratic behaviour or unexpected resets - this may be due to excessive current draw. Each LED can draw up to 50mA at 5V and full light intensity. Also, the LEDs are bright and sometimes dazzling to work with. You can lower the emitted intensities (and current draw) of the LEDs by altering values in the software.

The hardware example shown here was obtained for the Biomaker Stage 2 kit from the TZX Five Star store on Aliexpress (<https://www.aliexpress.com/item/1005002680484101.html>). Many variations on this theme can be found at other component suppliers.





# Using WS2812 RGB LEDs

## XOD library

The standard XOD library **xod-dev/ws2812** can be used to control the addressable RGB LEDs in the supplied hardware. The library contains a number of useful nodes, including the ability to flood all of the LEDs with a single chosen colour (**ws2812-mono**), to define a pattern or sequence of colour and to fill particular LEDs with this, or to address individual LEDs. The XOD nodes provide code to handle all of the complicated timing and signals to the LED devices.

If the hardware is properly connected, the general guide for setup is:

1. Create a new XOD patch: In the XOD IDE, create a new patch for your project.
2. Add a ws2812 controller node: the library contains two nodes for simple control of NeoPixel LED strips. The **xod-dev/ws2812/ws2812-mono** node allows flooding with a single colour, while **xod-dev/ws2812/ws2812-pattern** allows loading of arbitrary patterns in conjunction with the **xod-dev/ws2812/pattern** node. Both allow animation of LED displays. The library provides example patches that illustrate the use of these different nodes.
3. Select the relevant **ws2812** node and set the required parameters in the Inspector panel. Set the PORT value to the digital pin you connected the LED strip's data pin to and set the LED\_COUNT value to the number of LEDs in your strip.
4. Add colour nodes: To set the colour of individual LEDs, use the **xod-dev/ws2812/pattern** node. Drag the node to your patch and connect colour values to the indices of the LEDs you want to control (C0 for the first LED, C1 for the second, and so on). Connect the PAT output pin to a **xod-dev/ws2812/ws2812-pattern** node. Otherwise a colour value can be directly connected to a **xod-dev/ws2812/ws2812-mono** node.
5. Add the animation nodes: To create an animation or update the colors of the LEDs, adjust the value of the SHFT input for the **xod-dev/ws2812/ws2812-pattern** node (which will have the effect of rotating the pattern around the ring), or adjust the colour values of either WS2812 display node.
6. Upload the program: Connect your microcontroller to your computer and upload the XOD patch to see the RGB LEDs in action.

Two examples are shown on the facing page. Both assume that the RGB LED ring contains 12 LED elements and is connected via D10.

The first floods all LEDs with the same colour, using the Grove Beginner Kit button to switch the LEDs on and off, and the potentiometer to control the hue of the emitted light. It also converts the HSL colour to RGB values using one of the standard XOD nodes (**to-rgb**), and displays the value on the LCD display, assuming it is connected.

The second example shows the definition of a static pattern consisting of a rainbow sequence of colours, set by constant values. Each can be set using the in-built XOD colour picker. The potentiometer on the Grove Beginner Kit is used to set a start location for the pattern, ranging from position 0 to 12 (i.e. all around the clock face, back to the start). There are more examples provided in the **xod-dev/ws2812** library to allow some experimentation.

### Further information:

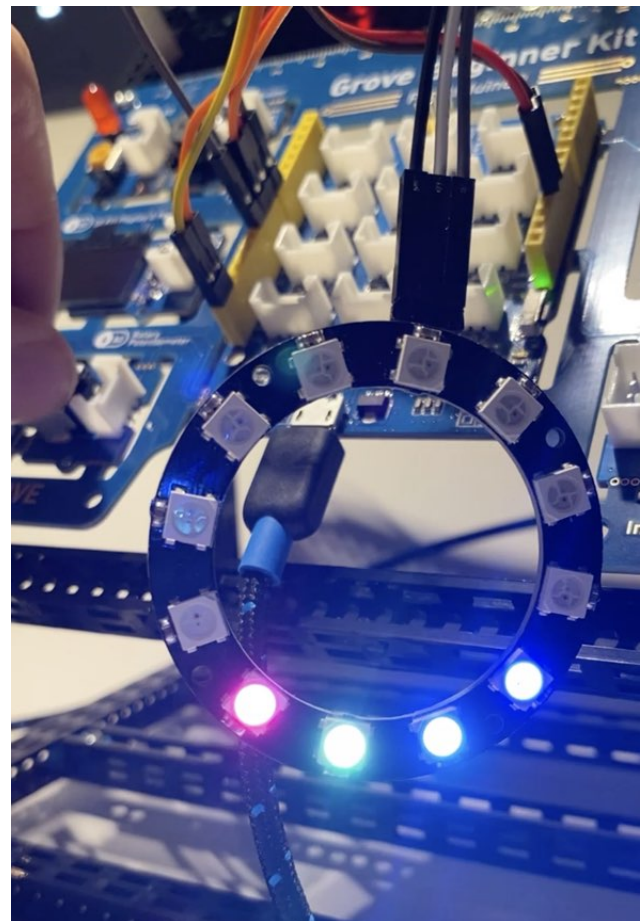
WS2812: guide to controlling NeoPixel RGB LEDs in XOD: <https://xod.io/docs/guide/ws2812-neopixel/>

### xod-dev/ws2812@0.37.3

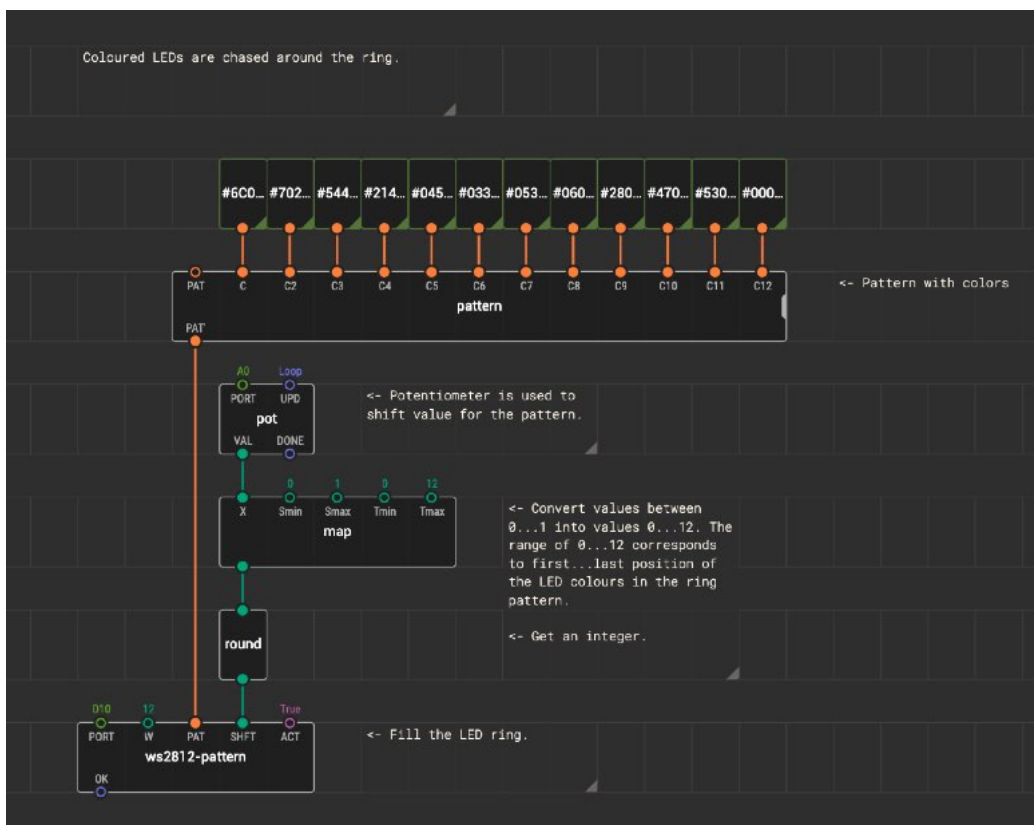
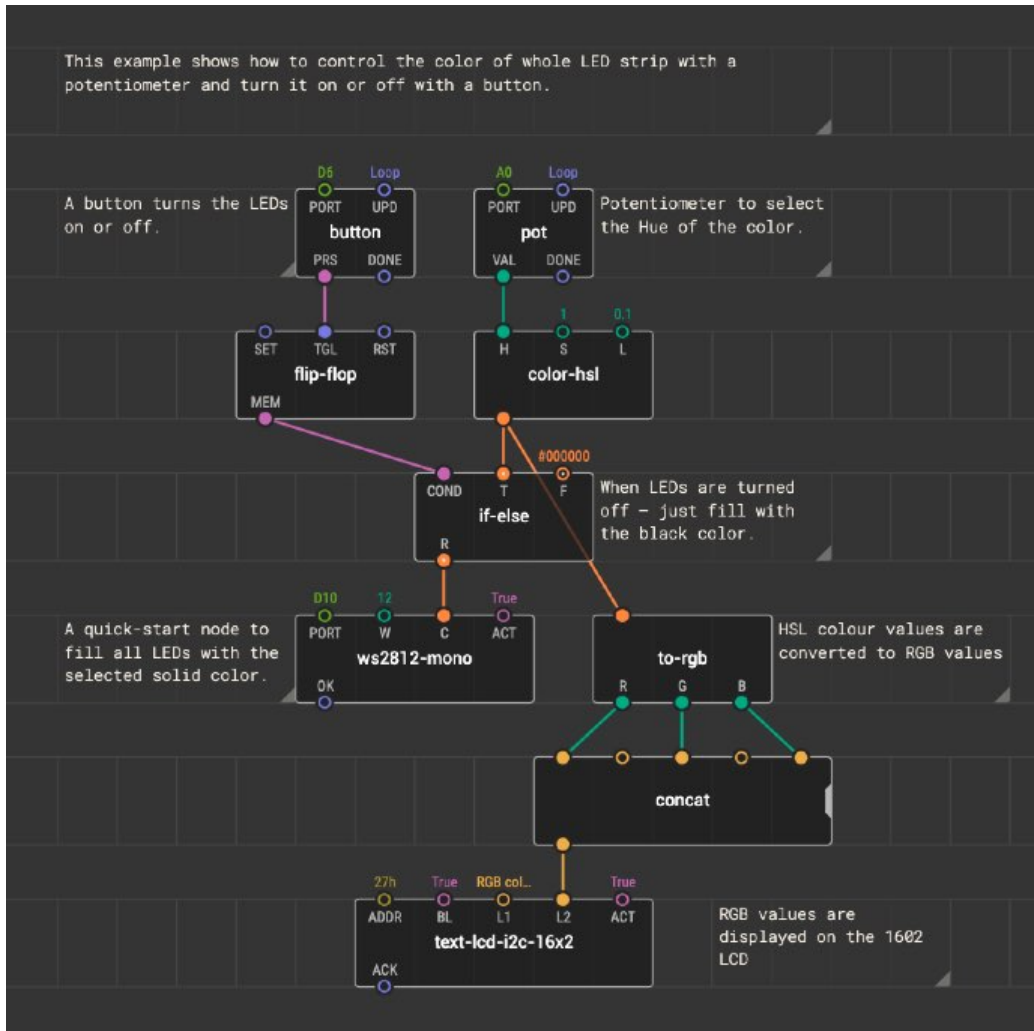
License: AGPL-3.0

Nodes to work with WS2812 (NeoPixel). The main difference from the other libraries that this library does not use a buffer to light up the LEDs. So it gives a possibility to light up a really long LED strip with a small microcontroller without a huge amount of RAM.

Node	Description
clear	Clears (turns off) of all pixels in the device.
example-fill-controlled	No description
example-fill-part	No description
example-pattern-crawling	No description
example-pattern-police	No description
fill-pattern	Fills the device LEDs with a repetitive color pattern.
fill-solid	Sets the color of pixels in the device to the same value.
pattern	Adds new colors to the tail of a given pattern.
ws2812-device	Represents a chained sequence of WS2812 pixels.
ws2812-pattern	No description
ws2812-mono	A quickstart node to control a LED strip which is always filled with a single color



## Test patches



# Real-time clock (DS1302)

## Description

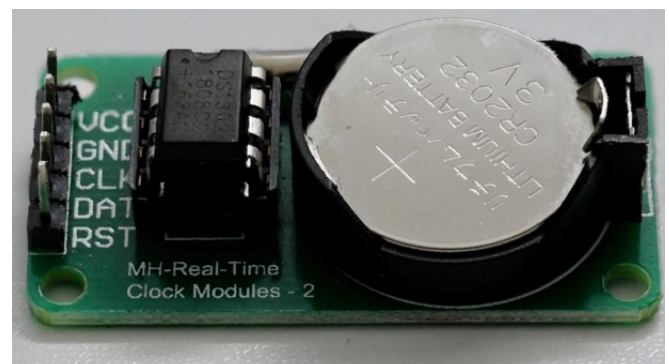
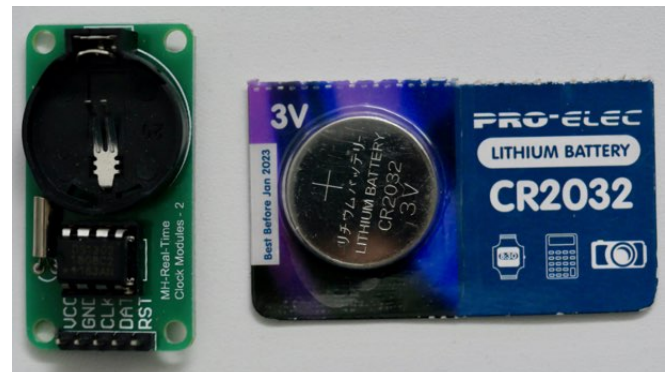
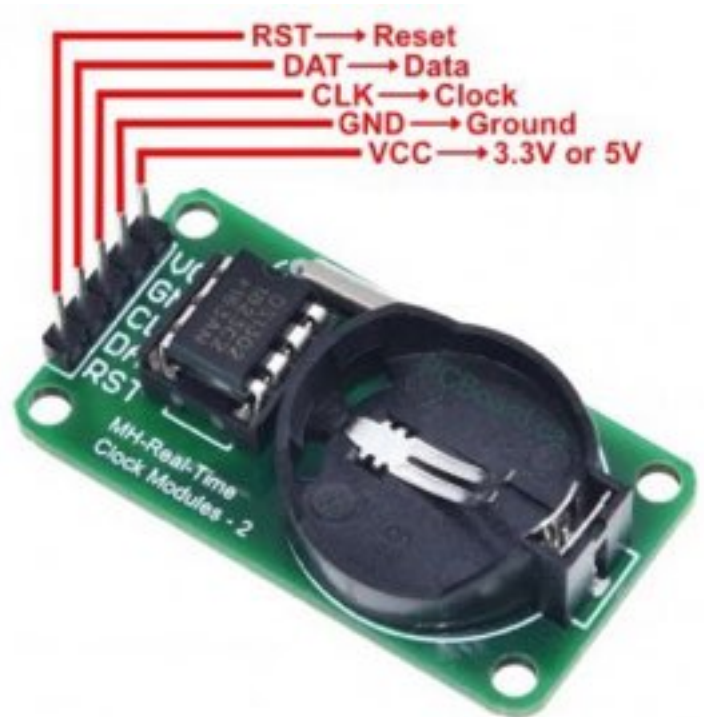
The DS1302 Real Time Clock (RTC) is a trickle-charge timekeeping chip contains a real-time clock and calendar and 31 bytes of static RAM and communicates with a microcontroller or microprocessor via a simple serial interface. The DS1302 real time clock and calendar provides seconds, minutes, hours, day, date, month, and year information. The date of the end of the month date is automatically adjusted, including corrections for leap year. The clock operates in a 24 hours or 12 hours format with AM and PM indicator. The module is very useful where it is important to control time-dependent events, or to record event timing.

The DS1302 is a Real-Time Clock (RTC) integrated circuit that keeps track of the current date and time, even when your main system or microcontroller is powered off. It is widely used in electronic devices, such as clocks, data loggers, and embedded systems, to maintain accurate timekeeping. The DS1302 uses a simple serial communication protocol called "3-Wire Interface" for communication with microcontrollers like Arduino.

Here's an overview of how the DS1302 RTC works:

- **Power supply:** The DS1302 requires a power supply (typically 3.3V or 5V) for its main operation and a small backup battery (usually a CR2032 coin cell) to keep the RTC running when the main power supply is disconnected. The backup battery ensures that the RTC maintains accurate timekeeping even during power outages or when the main system is turned off. These chips are designed to operate on very low power and retain data and clock information on less than 1 $\mu$ W.
- **Timekeeping registers:** The DS1302 has internal registers to store the current time and date information. These registers include seconds, minutes, hours, day of the week, date, month, and year. The DS1302 also supports 12-hour and 24-hour formats, with an option for automatic AM/PM indication in 12-hour mode.
- **3-Wire Interface:** The DS1302 communicates with microcontrollers using a simple serial protocol called the 3-Wire Interface. It requires three pins for communication: Clock (CLK), Data (DAT), and Chip Select (RST, CS or CE). The 3-Wire Interface is a bidirectional protocol, allowing data to be read from or written to the RTC.
- **Reading and writing data:** To read or write data to the DS1302, you need to send a command byte, followed by the data byte(s). The command byte specifies the register address and the operation (read or write). The data byte(s) contains the time or date information to be read or written.
- **Libraries and microcontroller support:** Various libraries and code examples are available for popular microcontrollers, such as Arduino and Raspberry Pi, making it easy to interface with the DS1302 RTC. For Arduino, you can use the "DS1302" library by Henning Karlsen, which provides functions for initializing the RTC, setting and reading the time and date, and more.

In summary, the DS1302 RTC works by maintaining a continuous record of the current date and time using its internal registers, powered by a backup battery when the main power supply is disconnected. The 3-Wire Interface allows communication with microcontrollers for reading and writing time and date data. DS1307 devices are also available, which have similar time-keeping specs (accurate to ~1 min per month at 25°C), but offer an I2C interface. Devices based on the DS3231 chip offer both I2C interface and higher accuracy (~2 min per year).





## How to Connect

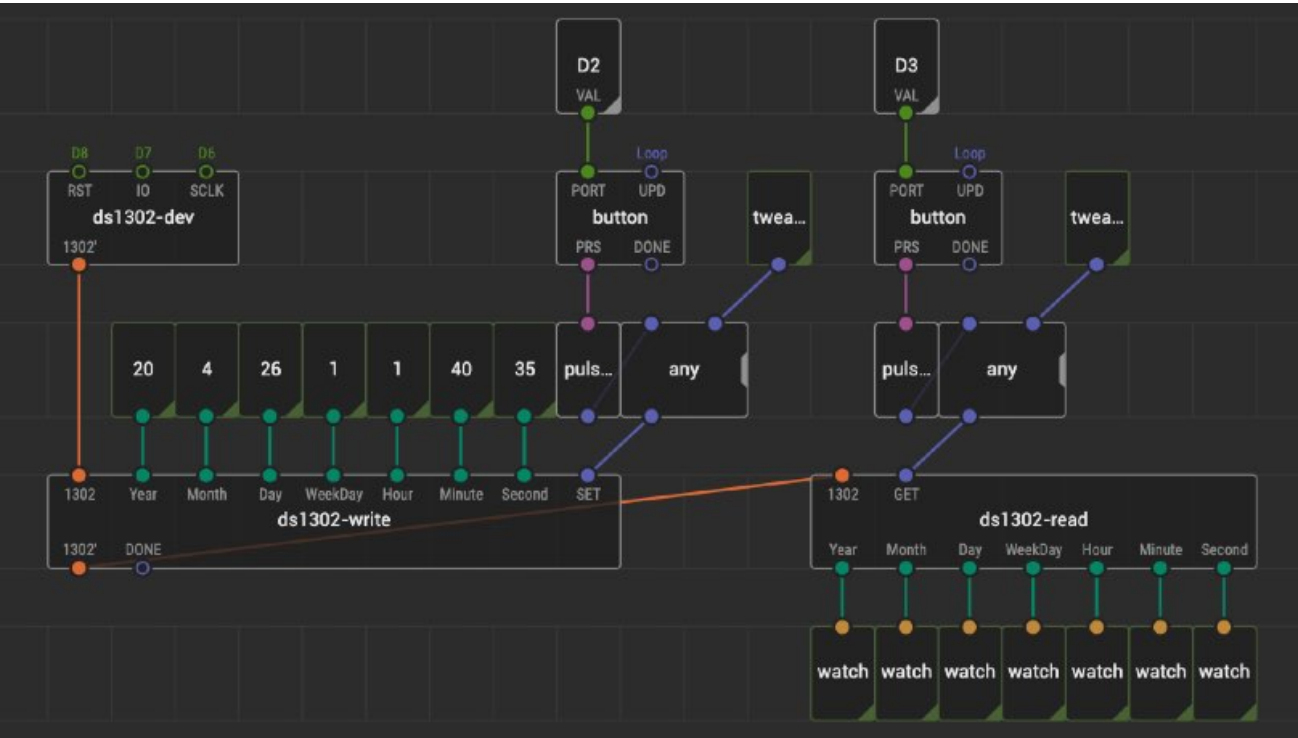
Important: the board is designed for use with a CR2302 coin cell lithium battery (3V) for backup of the set time and date. Install a battery if you require non-volatile storage. The estimated life of the battery will be up to 10 years.

Find details of the DS1302 RTC module used in the Biomaker Stage 2 kit here: <https://www.aliexpress.com/item/32833231512.html>

### To connect to the Grove board:

This board is set up to communicate with the Arduino micro controller via a 3-wire interface (rather than 2-wire I2C interface). There are 5 connecting pins on the module, including the power and ground pins, and these need to be connected to relevant ports on the central microcontroller board on the Grove Beginner Kit. You may connect pins on the DS1302 RTC board directly to the central yellow connector on the Grove Beginner Kit, or use the breadboard as an intermediate plug board. It is more convenient to use the breadboard if multiple components need to be powered at the same time.

- Select three unused digital ports on the microcontroller. For example, we will use ports D7, D8 and D9 in this example.
- Connect the three control pins (marked CLK, DAT and RST) to the digital sockets on the central microcontroller board on the Grove Beginner Kit - directly, or via the breadboard.
- Connect the GND pin/socket on the RTC module and microcontroller boards - directly, or via the breadboard.
- Connect the VCC (voltage common collector) pin on the RTC to a 5V source on the microcontroller board - again, directly or via the breadboard. This RTC module can be connected to 5V or 3.3V based microcontroller boards. (The Grove Beginner Kit uses 5V signals).



# Using the DS1302 RTC

## XOD library

The DS1302 clock can be obtained in different versions with either I2C or 3-wire communication protocols. The Stage 2 Biomaker kit contains one of the latter, and the **cesars/ds1302/ds1302** library should be loaded and used with this board.

The ds1302 library provides a base node for control of the RTC device (**ds1302-dev**), and this can be connected to **ds1302-write** and **ds1302-read** nodes as required. In order to use the RTC, you should set the current date-time the first time you use the clock. The write node is used for this (example shown right). The time is then updated automatically while the RTC is powered or connected to the backup battery.

The read node can then be triggered to obtain the current time when required. This provides a readout of the current year, month, day, weekday, hour, minute and second. The output can be formatted for display, creation of a timestamp or used for calculations.

## Further information:

Tutorial information: <https://forum.xod.io/t/how-to-read-time-from-non-i2c-ds1302-rtc-module/4341>

Maxim DS1302 data sheet: (<https://datasheets.maximintegrated.com/en/ds/DS1302.pdf>)

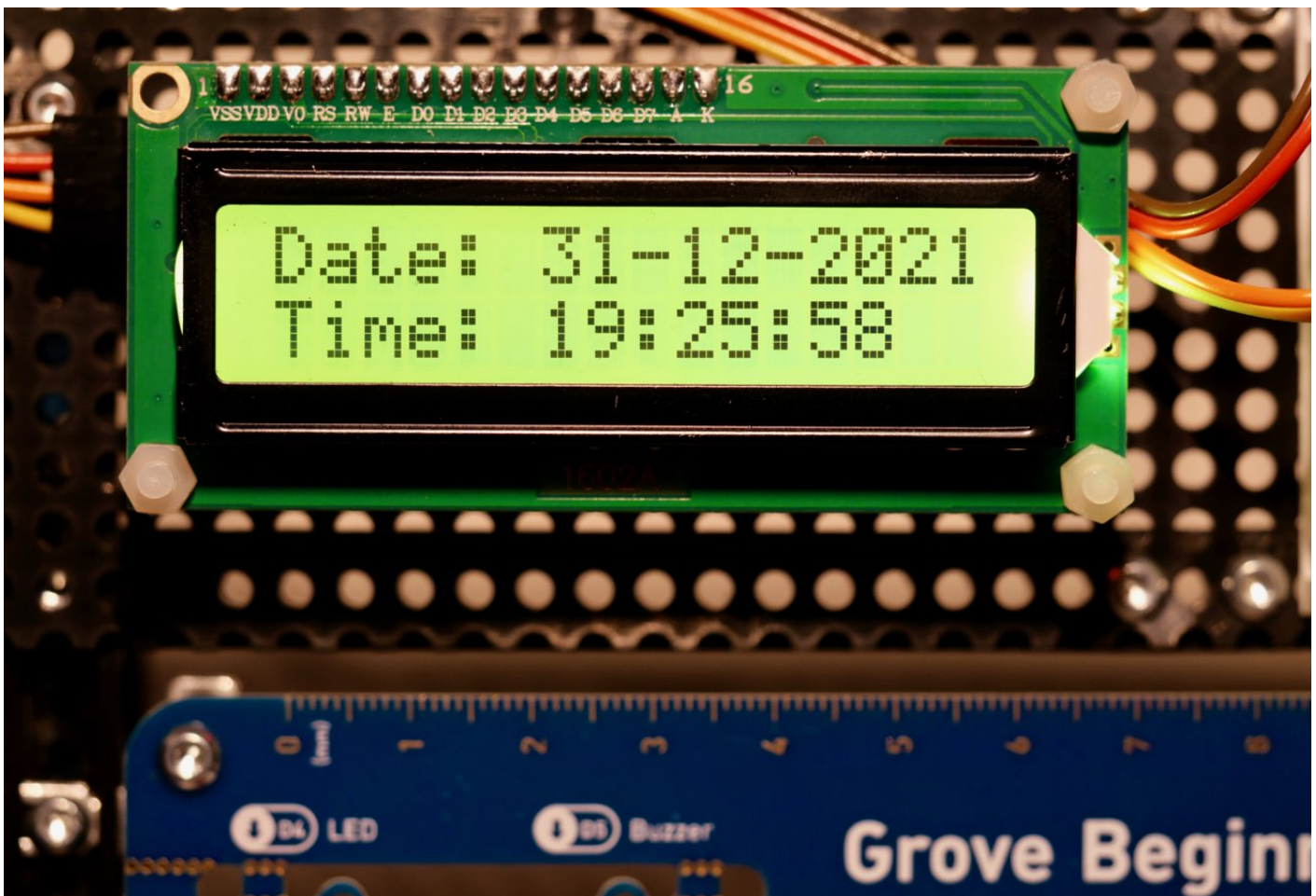
## cesars/ds1302@0.0.8

Library for RTC DS1302 ..... testing

Node	Description
<a href="#">00-readme</a>	No description
<a href="#">ds1302-dev</a>	No description
<a href="#">ds1302-read</a>	No description
<a href="#">ds1302-write</a>	No description

### Note for I2C RTC devices:

(XOD provides an in-built library `xod-dev/ds-rtc` for use with I2C RTC boards. The XOD website also provides a tutorial on how to use an I2C real-time clock module in XOD - <https://xod.io/docs/guide/rtc-example/> - including examples for formatting and handling dates in XOD.



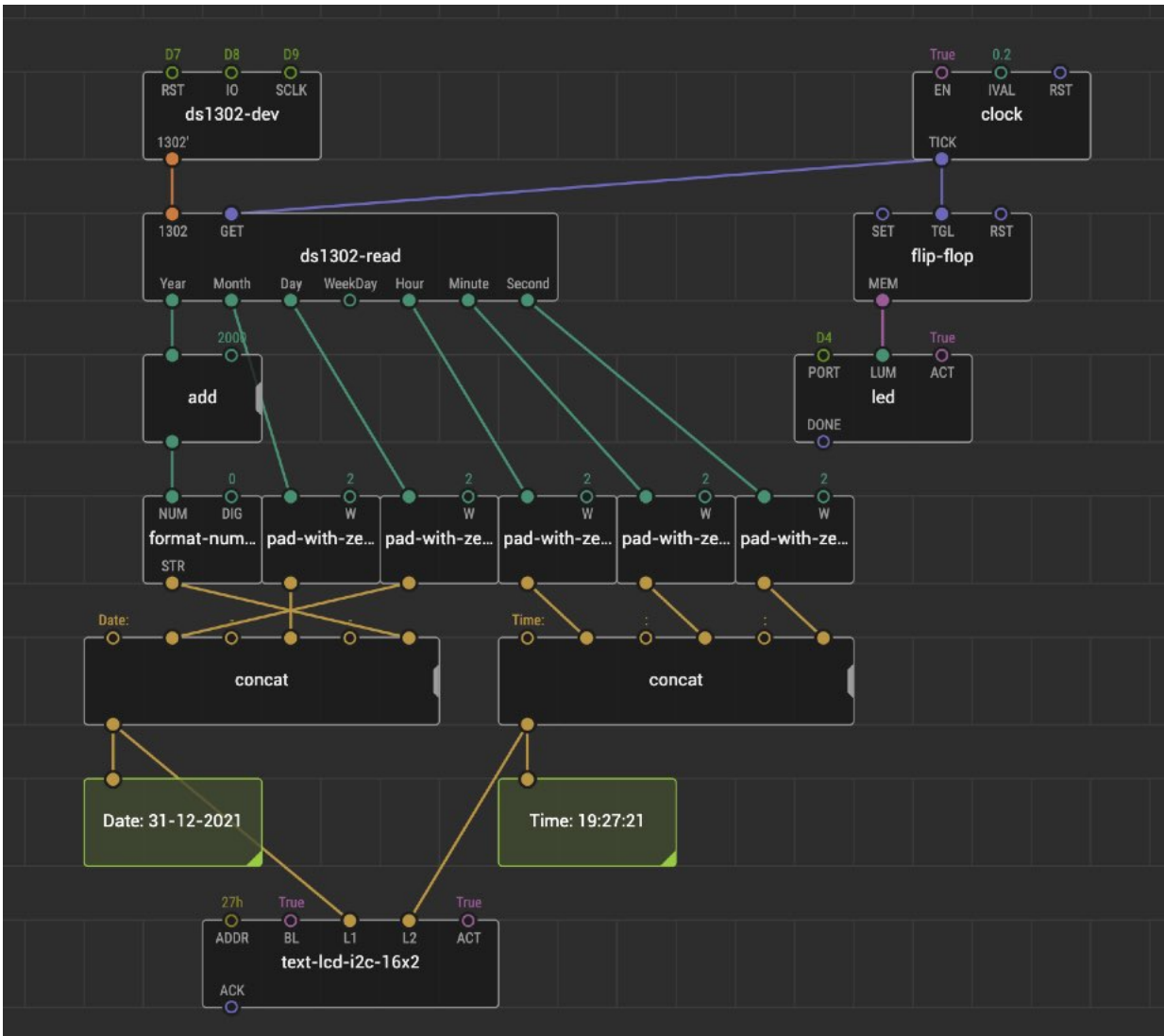




## Test patches

Example 1: Connect the DS1302 RTC board to ports D7, D8 and D9. One tweak node is provided to pulse the circuit, and write timing parameters to the chip, once to initialise the chip. A second tweak node triggers the reading of the RTC chip when required.

Example 2: A clock pulses every 200mSec, which triggers an LED to flip between states, and triggers reading of the RTC chip. The numerical output of the RTC is formatted for display on a 1602 LCD display - leading zeroes are padded, and suitable labels and spacer characters are added before being sent to the display. (see facing page)



# Temperature & Humidity Sensor (SHT20)

## Description

The SHT20 is a digital humidity and temperature sensor developed by Sensirion. It combines a capacitive humidity sensor and a bandgap temperature sensor on a single chip, providing highly accurate and reliable measurements. The SHT20 is often used in various applications, such as weather stations, HVAC systems, and environmental monitoring. Here's an overview of how the SHT20 sensor works:

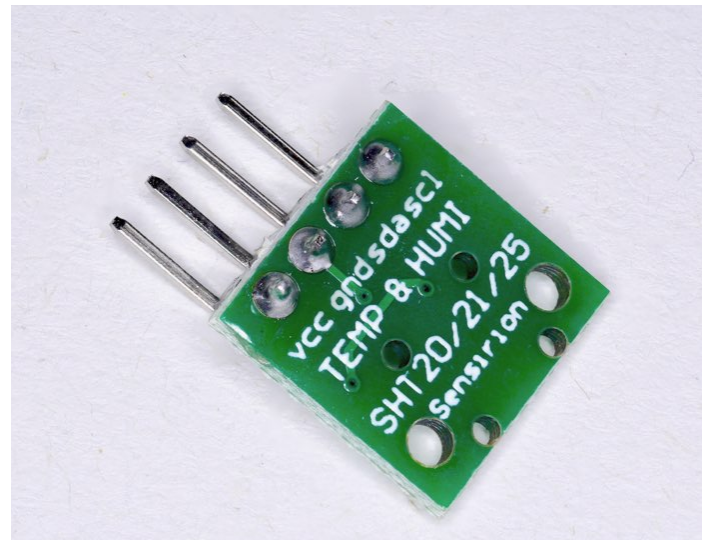
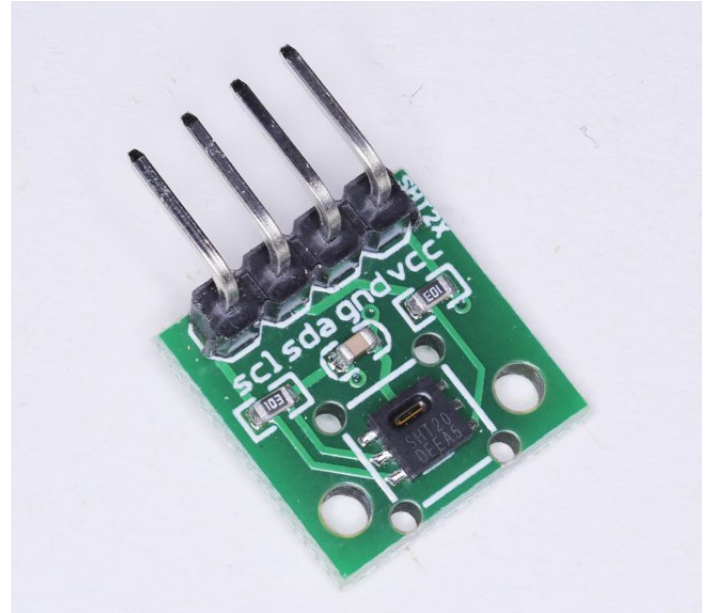
- 1. Capacitive humidity sensor:** The SHT20 measures relative humidity using a capacitive sensing element. This element consists of a thin polymer film sandwiched between two electrodes. As the humidity around the sensor changes, the dielectric constant of the polymer film changes, causing a variation in the capacitance between the electrodes. The sensor measures this capacitance change to determine the relative humidity.
- 2. Bandgap temperature sensor:** The SHT20 measures temperature using a bandgap temperature sensor. This type of sensor exploits the temperature dependence of the voltage across a p-n junction in a semiconductor material. As the temperature changes, the voltage across the p-n junction changes, which the sensor measures and converts into a temperature reading.
- 3. Signal processing and calibration:** The SHT20 sensor integrates an analog-to-digital converter (ADC) and a digital signal processor (DSP) on-chip. The ADC converts the analog signals from the capacitive humidity sensor and the bandgap temperature sensor into digital values. The DSP processes these digital values, applies calibration data stored in the sensor's memory, and calculates the final temperature and humidity readings.
- 4. Communication interface:** The SHT20 communicates with microcontrollers or other host devices using the I2C communication protocol. It supports standard and fast I2C modes, with a selectable 7-bit slave address. The sensor can be programmed to operate in different measurement resolutions and provides commands for initiating single-shot measurements or periodic data acquisition.
- 5. Power supply:** The SHT20 operates with a supply voltage range of 2.1V to 3.6V, making it suitable for various applications, including battery-powered devices. The sensor also features low power consumption and an optional on-chip heater that can be used for sensor self-testing or to prevent condensation in high-humidity environments.

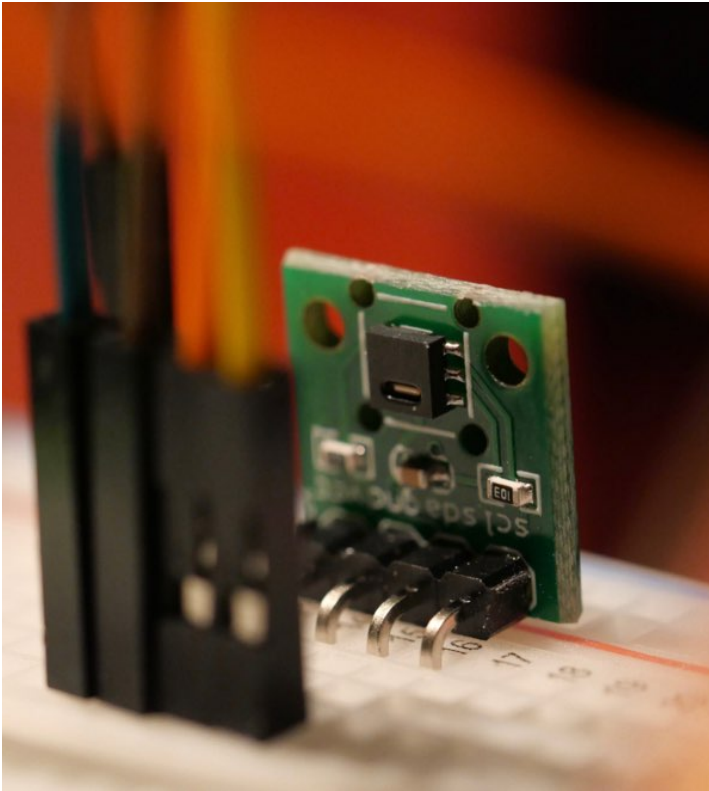
They come in a plastic package which leaves the sensor area open, protecting the devices from external impact and providing long-term stability. The SHT2x sensors are ideal for both high and low volume applications (SHT20 - Low cost, SHT21 - Standard, SHT25 - High Accuracy).

The devices are fully calibrated, and directly provide final measurements as a digital value. The Stage-2 Biomaker kit contains a sensor on a breakout board, ready for connection to the Grove Beginner Kit board, or equivalent.

Measurement parameters for the Sensirion SHT20 sensor.

- Temperature range: -40 to 125 C(-40 to 257 F)
- Humidity range: 0 to 100 % RH
- Temperature accuracy:  $\pm 0.5\%$  C
- Humidity accuracy:  $\pm 5\%$  RH
- Interface: I2C
- Voltage range: 2.1 - 3.6 V





## How to Connect

The module should be directly plugged into the breadboard, and adjacent sockets can be used to plug in Dupont cables to carry signals to/from the yellow-coloured header on the Grove Beginner Kit board. There are only four pins that need to be hooked up:

**VCC: Module power supply** - connect to 3.3V pin on the yellow-coloured header on the Grove Beginner Kit board.

**GND: Ground** - connect to a ground pin on the yellow-coloured header on the Grove Beginner Kit board.

**SDA: Serial Data Input/Output for the I2C protocol** - connect to SDA socket on the yellow-coloured header on the Grove Beginner Kit board.

**SCL: Serial Clock Input for I2C the protocol** - connect to SCL socket on the yellow-coloured header on the Grove Beginner Kit board.

(Alternatively, a Grove breadboard could be used, which can be directly connected to a I2C Grove socket).

While the Grove Beginner Kit board runs with 5V logic levels, the SHT-20 sensor board runs at 3.3V. Be sure to power the board from the 3.3V pin. Because I2C is an open drain signal, one can work without using electronic level shifters to convert the signal. In practice, the 3.3V signals of the SHT-20 device are adequate to communicate with the Arduino and tolerated by the the SHT-20. However, be aware that this is not a recommend procedure, and voltage compatible I2C buses or level converters should be used for permanent circuits. Meanwhile, this shortcut approach has proved stable enough for our prototyping with this device.

### justind000/ufire-sht20@0.9.0

License: MIT

Basic library to use the SHT2x Sensirion temperature and humidity sensors.

Node	Description
<a href="#">sht20</a>	Basic library to use the SHT2x Sensirion sensors.
<a href="#">sht20-example</a>	Demonstrates using the library to take a temperature and humidity measurement. Go to ufire.co for hydroponic related electrical components

### wayland/sht2x-rh-temp@0.0.2

License: MIT

Library for the Sensirion SHT2x series of humidity and temperature sensors. Wraps <https://github.com/RobTillaart/SHT2x>. Datasheet for SHT21: <https://sensirion.com/resource/datasheet/sht21>

Node	Description
<a href="#">sht2x-device</a>	Create an SHT2x device.
<a href="#">reset</a>	Soft reset.
<a href="#">get-identifiers</a>	Report the electronic IDs of the SHT2x device.
<a href="#">read</a>	Measure relative humidity and temperature.
<a href="#">hygrometer-thermometer</a>	Combines lower level nodes to create a ready to use sensor.
<a href="#">example-read-identifiers</a>	Patch for retrieving sensor identification numbers. Run in debug mode.
<a href="#">example-hygrometer-thermometer</a>	Patch for testing sensor. Run in debug mode.

## Choice of XOD libraries

There are two libraries available for use with the SHT20 sensors. They are both available from the xod.io site, and are **justind000/ufire-sht20** and **wayland/sht2x-rh-temp**. The first library provides simple, basic support for the SHT20 device, while **wayland/sht2x-rh-temp** provides wider and deeper support for this class of devices. For example, it provides access sensor identifiers, testing, error readings, etc.





# Using the SHT20 sensor

## XOD node for readout

For this example - the external library `justind000/ufire-sht20` should be imported into your XOD development environment. It provides the `sht20` node, which handles communication with the I2C device, and delivery of calibrated temperature and humidity values.

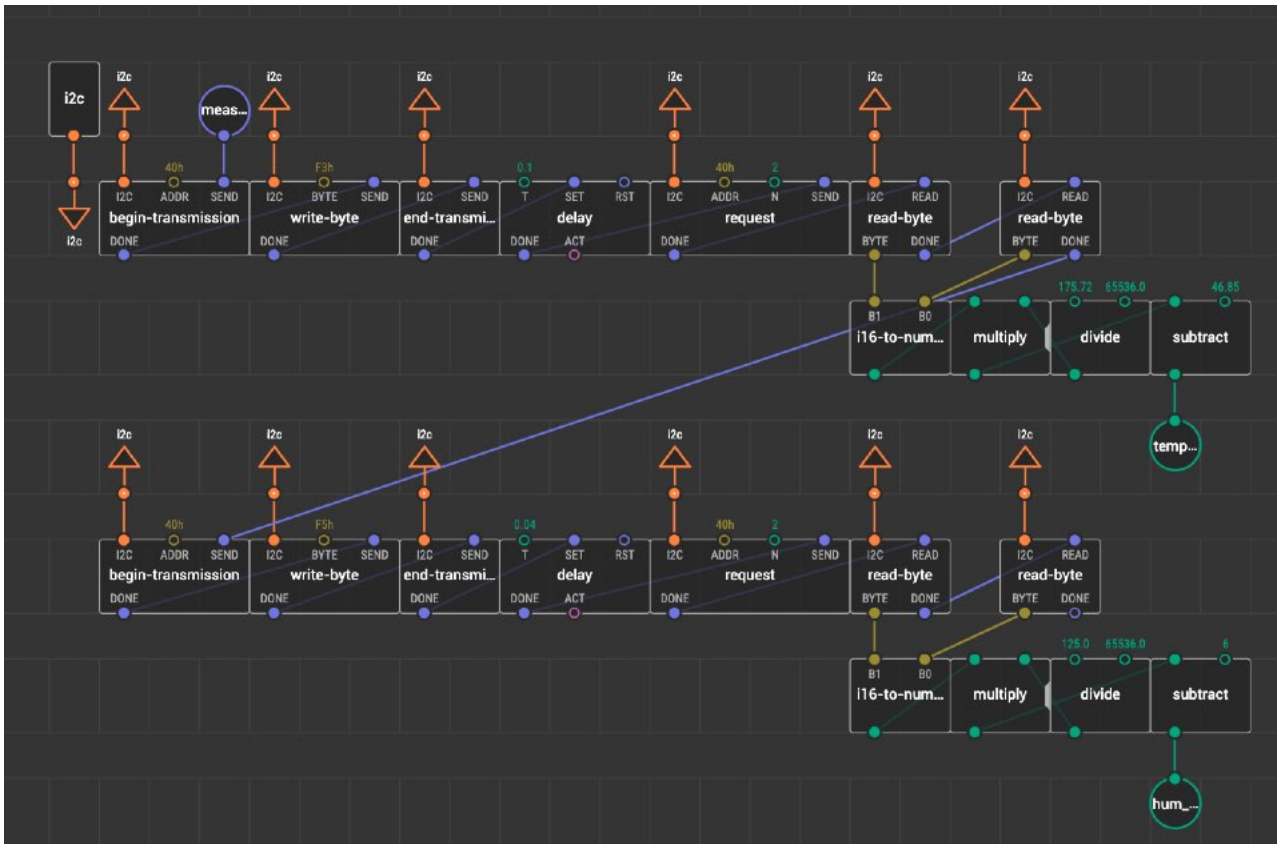
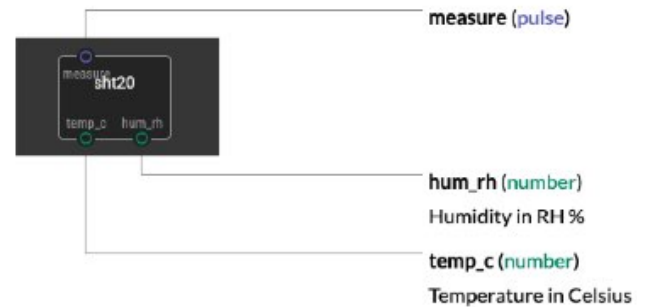
A pulse input is required to trigger each measurement, and after a short delay, the temperature and humidity values are delivered for subsequent display, or calculations.

The node is simple to operate, but contains complex inner working at the level of both hardware and software. The integrated hardware is outlined on the previous pages, and the operation of the node is shown below. The node is itself composed of other nodes from the `justind000/ufire-sht20` library that control polling and readout of the SHT20 device via the I2C serial communication protocol.

## sht20

`justind000/ufire-sht20/sht20`

Basic library to use the SHT2x Sensor on sensors.

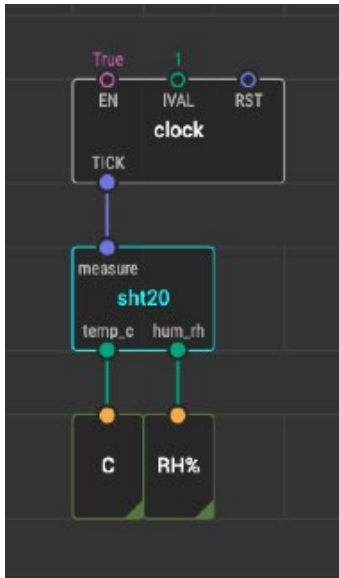


## Further information:

Product page: <https://www.sensirion.com/en/environmental-sensors/humidity-sensors/humidity-temperature-sensor-sht2x-digital-i2c-accurate/>

A wide range of related technical documents and application notes about the SHT-2x series can be found at: <https://www.sensirion.com/en/download-center/humidity-sensors/humidity-temperature-sensor-sht2x-digital-i2c-accurate/>

Experimental comparison of multiple humidity sensors: <https://wiki.liutyi.info/display/ARDUINO/Sensors>

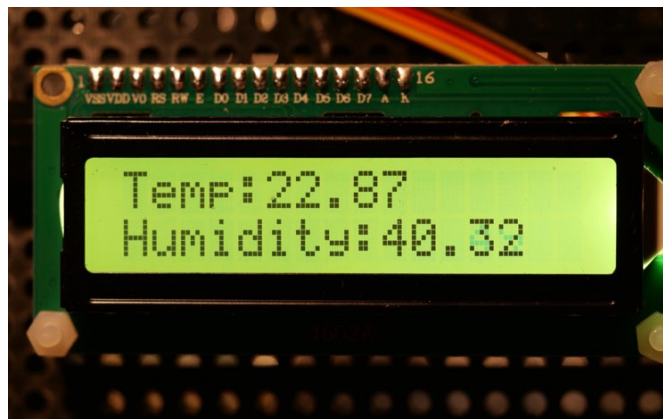
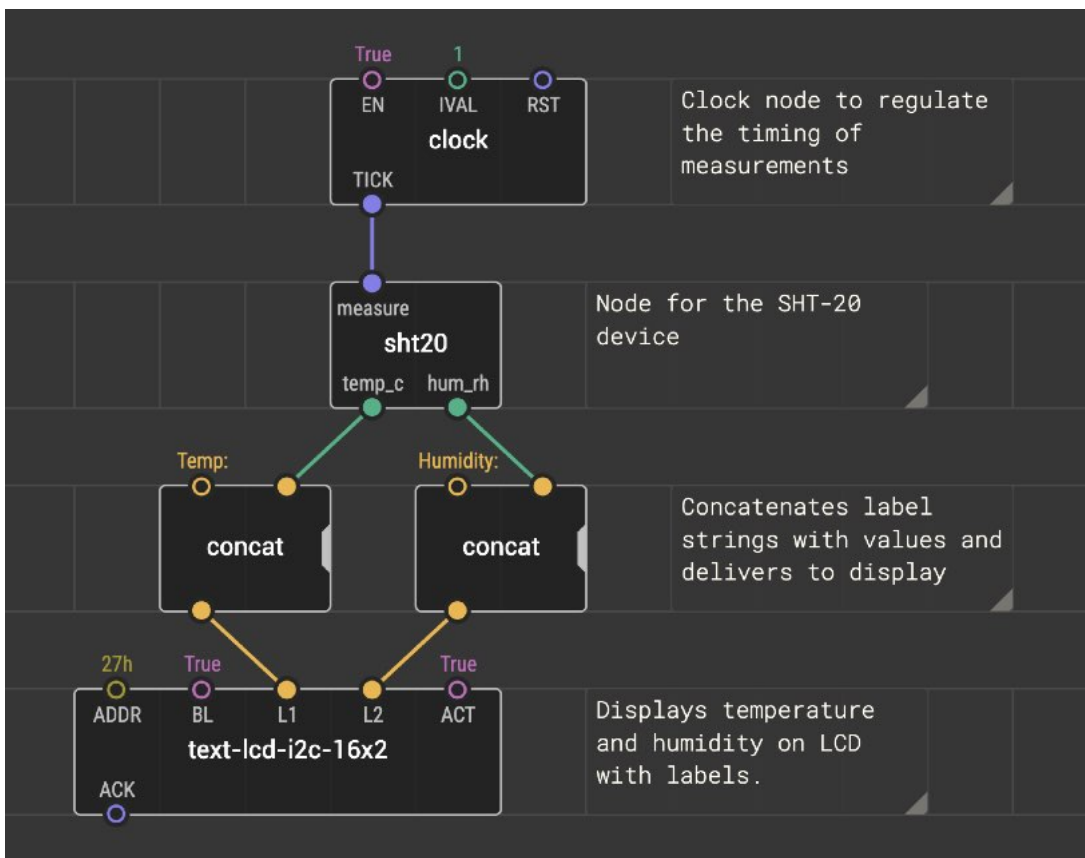


## Test patch

Simple patches can be used to test the connected device.

First, the **sht20** node can be connected to two watch nodes - and clock node can be used to trigger measurements at regular intervals. The read process includes a delay (~150 ms) so timing should be set to avoid oversampling. Here a 1 sec delay is included.

Second, the patch can be modified to show the temperature and humidity values on the 16x2 LCD display. Text labels can be added using concat nodes, and fed to the **text-lcd-i2c-16x2** node. The SHT-20 and 16x2 LCD should be connected to the Grove Beginner Kit board through the I2C bus.



# Calibrated Light Sensor (BH1750)

## Description

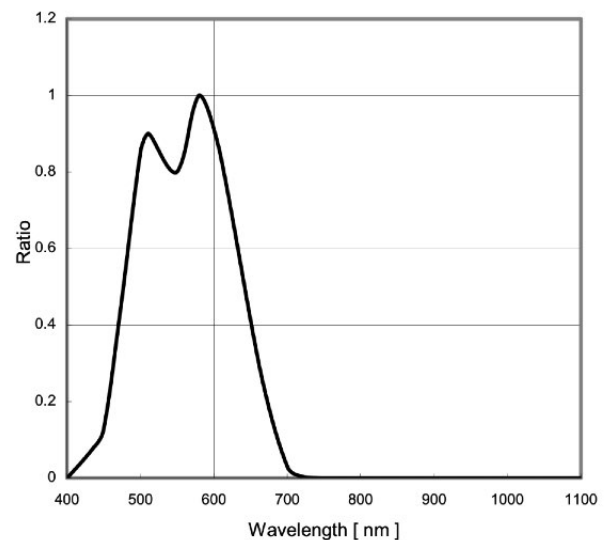
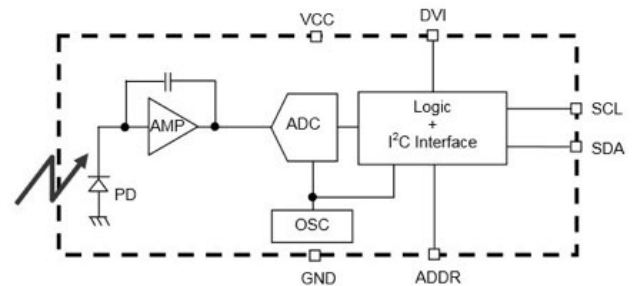
The BH1750 is a digital ambient light sensor that accurately measures illuminance (light intensity) in a wide range of lighting conditions. It is commonly used in applications like lighting control systems, display backlight control, and weather stations. The BH1750 integrates a photodiode, an analog-to-digital converter (ADC), and an I2C interface on a single chip, making it easy to interface with microcontrollers and other host devices.

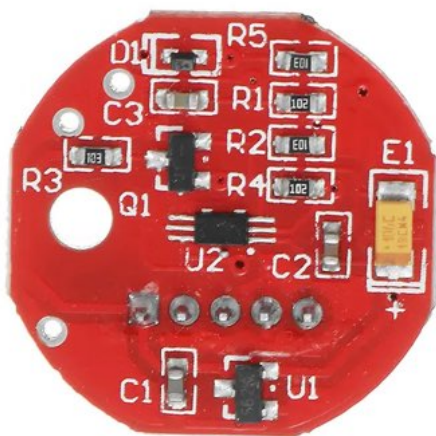
Here's an overview of how the BH1750 light sensor works:

- 1. Photodiode:** The core component of the BH1750 is a photodiode that is sensitive to light in the visible spectrum. The photodiode generates a photocurrent when it absorbs light. The amount of photocurrent produced is proportional to the light intensity (illuminance) falling on the sensor.
- 2. Spectral response:** The BH1750's photodiode has a spectral response close to the human eye's luminosity function, also known as the  $V(\lambda)$  curve. This means that the sensor's response to different wavelengths of light closely resembles the human eye's sensitivity, resulting in more accurate and reliable illuminance measurements.
- 3. Analog-to-digital conversion:** The photocurrent generated by the photodiode is converted into a digital value using an integrated analog-to-digital converter (ADC). The ADC resolution can be configured to 4, 8, or 16 bits, depending on the desired measurement accuracy and speed - from 1–65535 lux (lx)
- 4. I2C communication interface:** The BH1750 communicates with microcontrollers or other host devices using the I2C communication protocol. It supports standard and fast I2C modes and has a fixed 7-bit slave address. The sensor provides various commands for configuring the measurement mode, resolution, and timing, as well as reading the illuminance data.
- 5. Measurement modes:** The BH1750 supports multiple measurement modes, including continuous and one-time measurements. In continuous mode, the sensor periodically measures the light intensity and updates the illuminance data at regular intervals. In one-time mode, the sensor performs a single measurement and then automatically enters a low-power mode to conserve energy.
- 6. Power supply:** The BH1750 operates with a supply voltage range of 2.4V to 3.6V, making it suitable for various applications, including battery-powered devices. The sensor also features low power consumption and can be put into a low-power mode when not actively measuring illuminance.

The supplied device has a white plastic dome to provide a wide angle of detection. It can be powered as a 3-5V device, and has a built in electronics to allow direct digital output, bypassing additional calculation, and calibration. It has a response close to human visual sensitivity, across the visible spectrum.

The lux (symbol: lx) is the SI unit of illuminance and luminous emittance, measuring luminous flux per unit area. It is equal to one lumen per square metre. In photometry, this is used as a measure of the intensity, as perceived by the human eye, of light that hits or passes through a surface. It is analogous to the radiometric unit watts per square metre, but with the power at each wavelength weighted according to the luminosity function, a standardized model of human visual brightness perception.





### Typical Lux values

- 0.0001 lux - Moonless, overcast night sky (starlight)
  - 0.002 lux - Moonless clear night sky with airglow
  - 0.27–1.0 lux - Full moon on a clear night[3][4]
  - 3.4 lux - Dark limit of civil twilight under a clear sky
  - 50 lux - Family living room lights (Australia, 1998)
  - 80 lux - Office building hallway/toilet lighting]
  - 100 lux - Very dark overcast day
  - 320–500 lux - Office lighting
  - 400 lux - Sunrise or sunset on a clear day.
  - 1000 lux - Overcast day; typical TV studio lighting
  - 10000–25000 lux - Full daylight (not direct sun)
  - 32000–100000 lux - Direct sunlight
- (source: Wikipedia)

The outdoor range of light range is from < 1- 120,000 lux so the bare, un-covered sensor will fully saturate in full sun at 54612 lux if not adjusted or has a cover applied. In bright light, over-range values will have a negative sign. The readings can be compensated for in the software at higher light levels. For example, the supplied sensor "BH1750FVI Chip Light Intensity Light Module Light ball" has a white dome cover, and the sensor receives around 52 % of the actual light level when directly overhead.

### How to Connect

The BH1750 device communicates with the microcontroller across an I2C bus. It can be powered from a 3.3V-5V supply. The module is provided with a plug-in socket (XH2.54) and set of leads. A set of male-male Dupont leads can be used to connect the leads directly to the yellow-coloured main connector on the Grove Beginner Kit board - or indirectly, via sockets on the supplied breadboard.

- The SCL and DAT leads from the BH1750 module should be connected to the SCL and SDA sockets, respectively
- The GND lead should be connected to a ground socket
- The Vcc lead should be connected to a 5V socket
- The ADDR pin can be used to select the device if you wish to use multiple light detectors.





# Using the BH1750 light sensor

## XOD library

Download the external XOD library: [vitaliysh/bh1750](#). This provides a single node, **bh1750**, that can be used to provide readings from the sensor. The node requires the I2C address of the device (23h) and provides the measured light intensity as a number. Reads can be triggered by sending pulses to the node, and the author of the node recommends maintaining a gap between readings. There is some more discussion of the library on the XOD forum at: <https://forum.xod.io/t/my-node-gy-302-bh1750-light-sensor-i2c/2124>

## Further information:

Building PAR light meters with different digital sensors, including the BH1750: <https://ledgardener.com/forum/viewtopic.php?t=5748>

BH1750 Light Sensor Practical notes- Problems and issues: <http://community.heltec.cn/t/bh1750-light-sensor-practical-notes-problems-and-issues/1521>

BH1750 Ambient Light Sensor Interfacing with Arduino: <https://microcontrollerslab.com/bh1750-interfacing-with-arduino-measure-light/>

BH1750 Datasheet: <https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf>

Other comparable I2C light sensors are the:

**AMS-TAOS TSL2561 and TSL2591** (higher sensitivity/accuracy) which have dual visible and infrared photodiodes.

**Broadcom APDS-9301** with integrated photodiode, ADC, and I2C interface on a single chip.

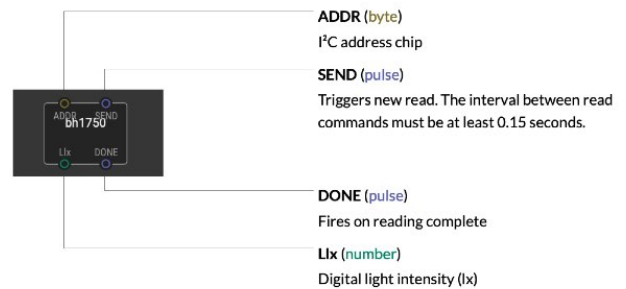
**Vishay VEML7700** with photodiode and an ADC with multiple gain settings.

**Maxim Integrated MAX44009** with ultra-wide dynamic range and very low power consumption. This is of particular interest for outdoors use in bright sunlight, as the sensor can measure up to 188,000 lux, and a XOD library is available at: [denis-nabatchikov/max-44009](#)

## bh1750

[vitaliysh/bh1750/bh1750](#)

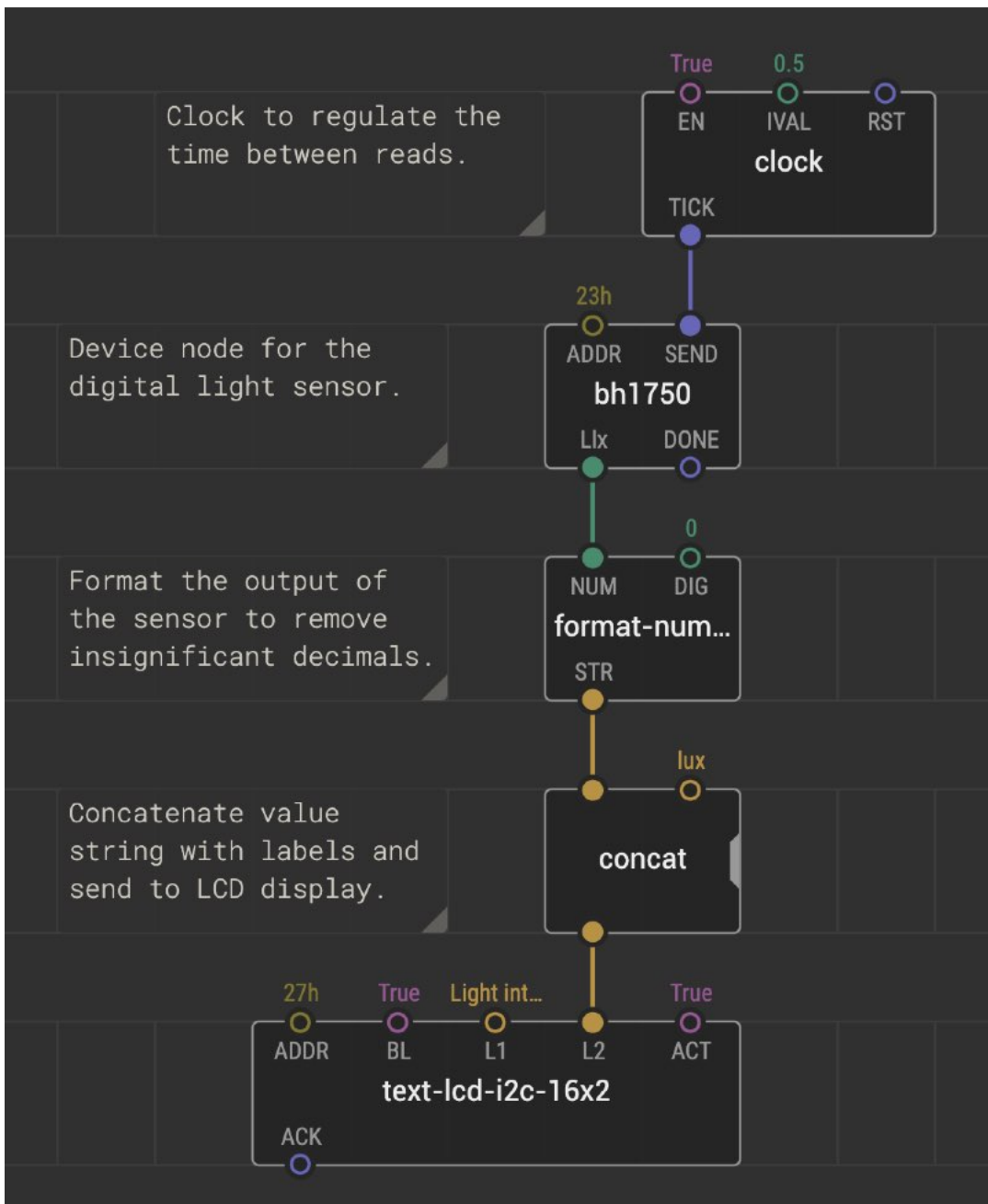
GY-302 BH1750 is a digital Ambient Light Sensor IC for I2C bus interface. (1 - 65535 lx)





## Test patch

A working test patch is shown below. This uses a **clock** node to generate 100 millisecond-spaced pulses to read light values from the BH1750 device. The output from the **bh1750** node is fed to the **format-number** node, to remove insignificant decimal values, and produce a string for display. A **concat** node is used to add the " lux" label, and sent to the **text-lcd-i2c-16x2** node for display on the LCD - which is also attached to the I2C bus. The first line of the display shows the label: "Light intensity:" The patch provides a real-time display of light intensity as it plays across the sensor. Images below show the light measurements taken under ambient room light (168 lux), and after illumination with a hand-held torch (7980 lux).



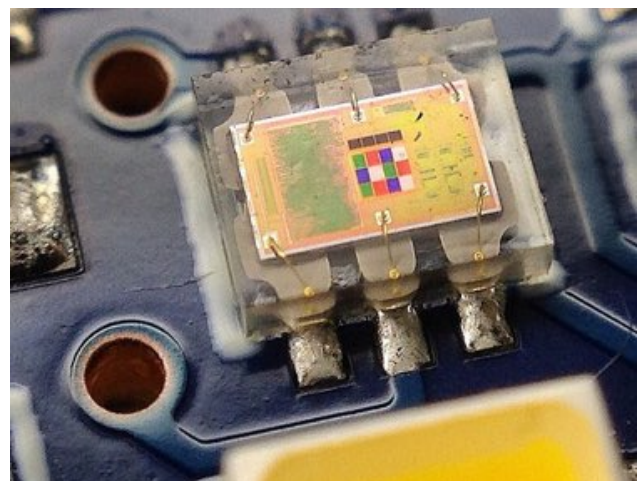
# Colour sensor TCS3472

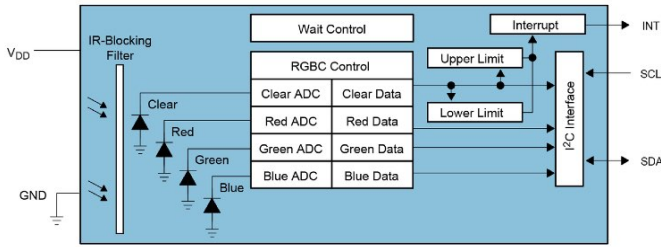
## Description

The TCS3472 and TCS34725 are digital color light-to-digital converters developed by AMS-TAOS. They measure the intensity of red, green, blue (RGB), and clear (unfiltered) light, enabling it to detect a wide range of colors and ambient light levels.

- 1. Photodiodes:** The TCS3472 integrates four types of photodiodes on a single chip: red, green, blue, and clear (unfiltered). Each photodiode type is sensitive to a specific wavelength range, allowing the sensor to detect the intensity of each colour channel (RGB) and clear light independently.
- 2. Colour filters:** Each colour photodiode (red, green, and blue) is covered with a colour filter that selectively allows light of specific wavelengths to pass through. These filters ensure that the photodiodes respond primarily to the desired colour channel, providing accurate colour measurements.
- 3. Analog-to-digital conversion:** The photocurrent generated by each photodiode is converted into a digital value using an integrated analog-to-digital converter (ADC). The ADC resolution can be configured to 12 or 16 bits, depending on the desired measurement accuracy and speed.
- 4. I2C communication interface:** The TCS3472 communicates with microcontrollers or other host devices using the I2C communication protocol. It supports standard and fast I2C modes and has a selectable 7-bit slave address. The sensor provides various commands for configuring the measurement mode, resolution, gain, and timing, as well as reading the colour and clear light data.
- 5. Programmable gain and integration time:** The TCS3472 allows users to configure the gain and integration time settings. The gain setting determines the sensor's sensitivity, while the integration time setting controls the duration of light measurement. Adjusting these settings enables the sensor to operate effectively in various lighting conditions and optimize the measurement accuracy and dynamic range.
- 6. Interrupt function:** The TCS3472 features an interrupt function that can be configured to trigger when the measured light intensity falls outside a specified range, exceeds a certain threshold, or changes by a specific amount. This feature allows the host device to notify the host device of significant changes in the ambient light, reducing the need for continuous polling and saving power.
- 7. Power supply:** The TCS3472 operates with a supply voltage range of 2.7V to 3.6V, making it suitable for various applications, including battery-powered devices. The sensor also features low power consumption and a power-down mode to conserve energy when not actively measuring light.

Suitable modules with a working voltage of 3.3V/5V are available from Aliexpress (e.g. <https://www.aliexpress.com/item/1005004525849112.html>).





## How to Connect

The TCS3472 Color Sensor board has 7 pins:

**VIN:** Module power supply of 5 V

**GND:** Ground

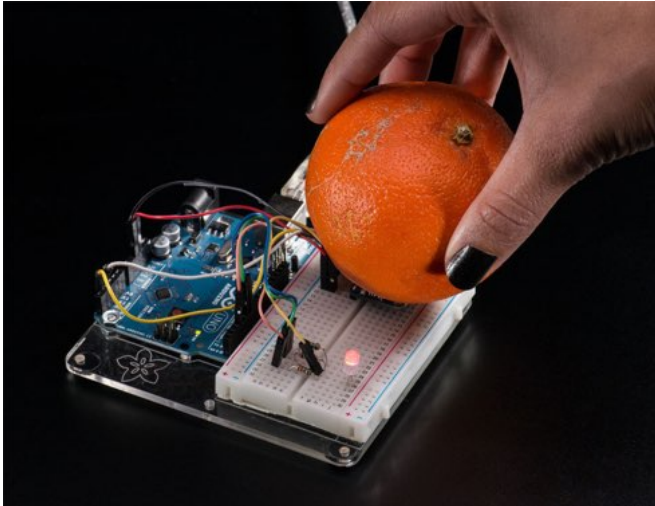
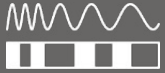
**3V3:** Low voltage module power supply of 3.3 V

**SCL:** I2C Clock

**SDA:** I2C data

**INT:** Adjust I2C Address

**LED:** Turning on the LED

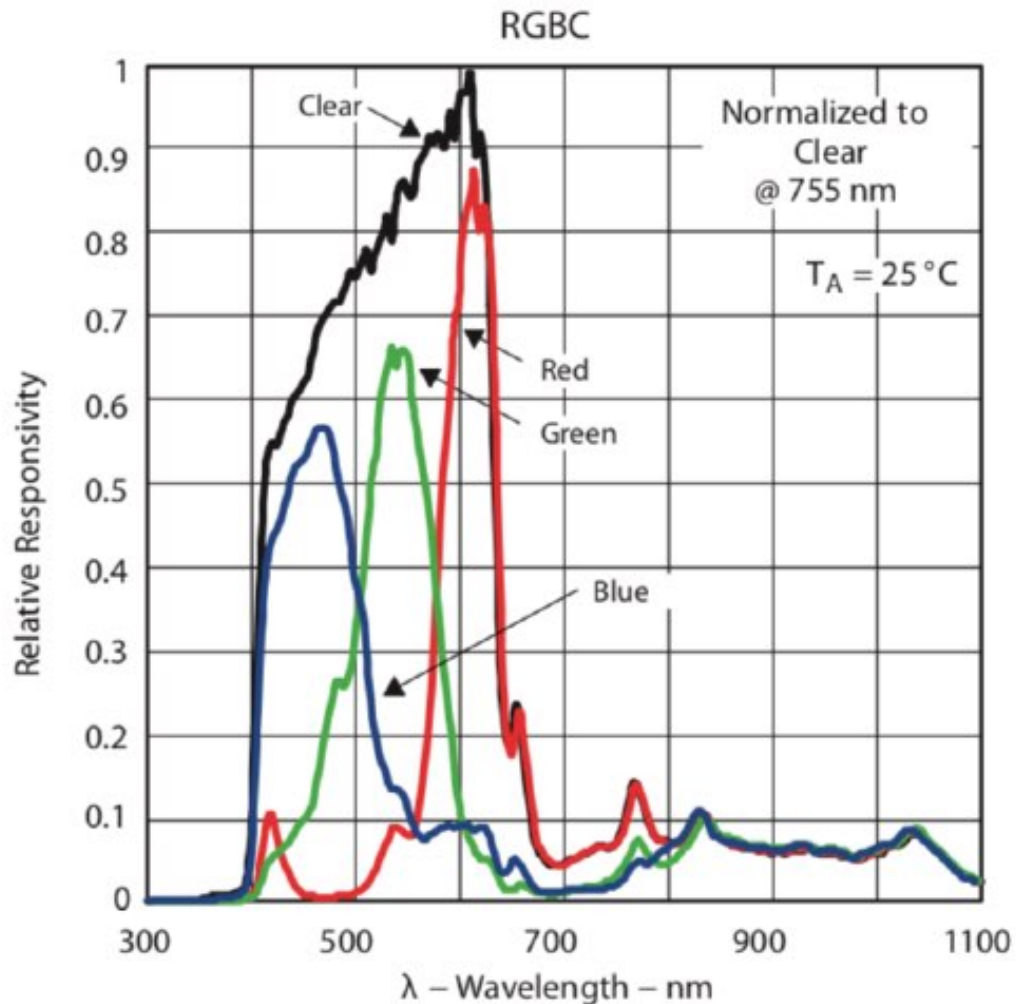


To connect to the Grove board:

- Connect four wires to pins inside one of the I2C sockets on the central portion of the board (top right)
- Connect left-most wire (GND) to GND pin on module
- Connect the second-left wire (VCC) to VCC pin on module
- Connect the second-right wire (SDA) to SDA pin on module
- Connect the right-most wire (SCL) to SCL pin on module

(or alternatively connect directly via hookup wires and breadboard to ports on the Grove Beginner board)

When connected to the Arduino I2C bus, XOD patches can read off a series of calibrated values including: the intensity of light in the blue part of the spectrum (465nm), green (525nm) and red (615nm). In addition, the intensity of unfiltered clear light, illuminance value in lux and colour temperature (degrees K) can be read out.



# Using a TCS3472 colour sensor

## XOD library

First, the external XOD library **antoniorg/tcs34725** should be loaded in the XOD environment. This library provides nodes and example patches that allow operation of the sensor. The contents of the library is shown below, and a node for reading the TCS3472 sensor is shown right.

Note that the library supports programmable switching on/off of the onboard white LED. This is useful for measuring the spectral properties of reflective materials.

## Further information:

Colour recognition with the TCS3472: <https://www.hackster.io/t3486784401/color-recognition-piano-62a16e>

Characterisation of soil properties using a simple colour detection device: <https://www.hackster.io/antonio-ruiz/characterisation-of-soil-properties-using-a-simple-device-6c9e9b>

Professional Hydroponics Light Monitoring: <https://www.hackster.io/chuygen/professional-hydroponics-light-monitoring-3547dd>

Multiplexing 6 I2C TCS34725 Color Sensors: <https://www.hackster.io/sherwinchiu89/multiplexing-6-i2c-tcs34725-color-sensors-2a7272>

Technical description for measuring color temperature with the TCS3472: <https://circuitcellar.com/research-design-hub/projects/white-hot-measuring-color-temperature/>

Download technical documents: <https://ams.com/en/tcs34725#tab/documents>

Latest products in the TCS line: <https://ams.com/en/color-sensors>

Other I2C RGB colour sensors that may be worth exploring are the:

AMS-TAOS TCS3200 color sensor

Broadcom APDS-9960

ROHM Semiconductor BH1745

Vishay VEML6040

In addition, AMS supply a range of I2C multi-spectral light sensors that can provide 6-18 channel detection across visible and adjacent wavelengths (<https://ams.com/en/spectral-sensing>).

## antoniorg/tcs34725@0.0.3

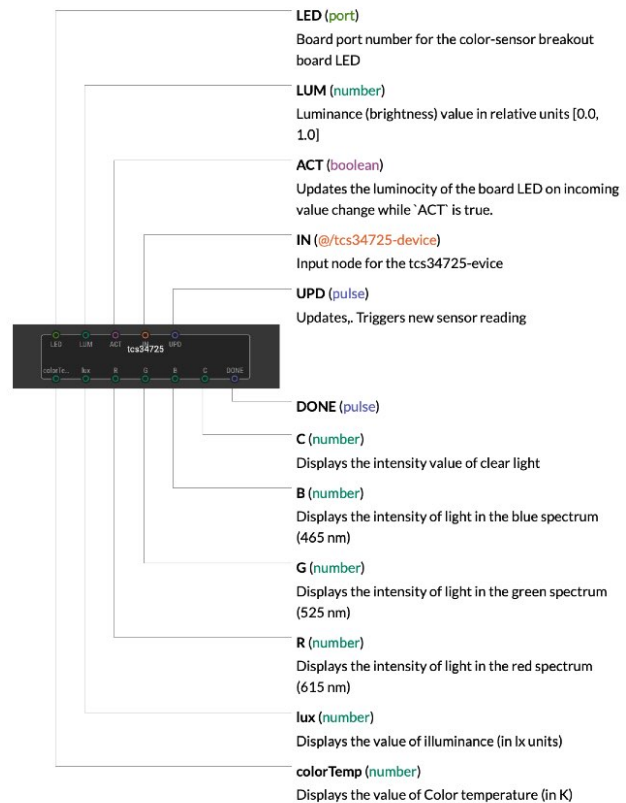
Library for the Adafruit RGB Color Sensor tcs34725

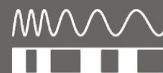
Node	Description
<a href="#">tcs34725-device</a>	Device node connected to the tcs34725-raw-data
<a href="#">tcs34725-raw-data</a>	No description
<a href="#">tcs34725</a>	Node to get the data from tcs3475, but the LED is continuously on
<a href="#">example-tcs3475</a>	No description

## tcs34725

[antoniorg/tcs34725/tcs34725](#)

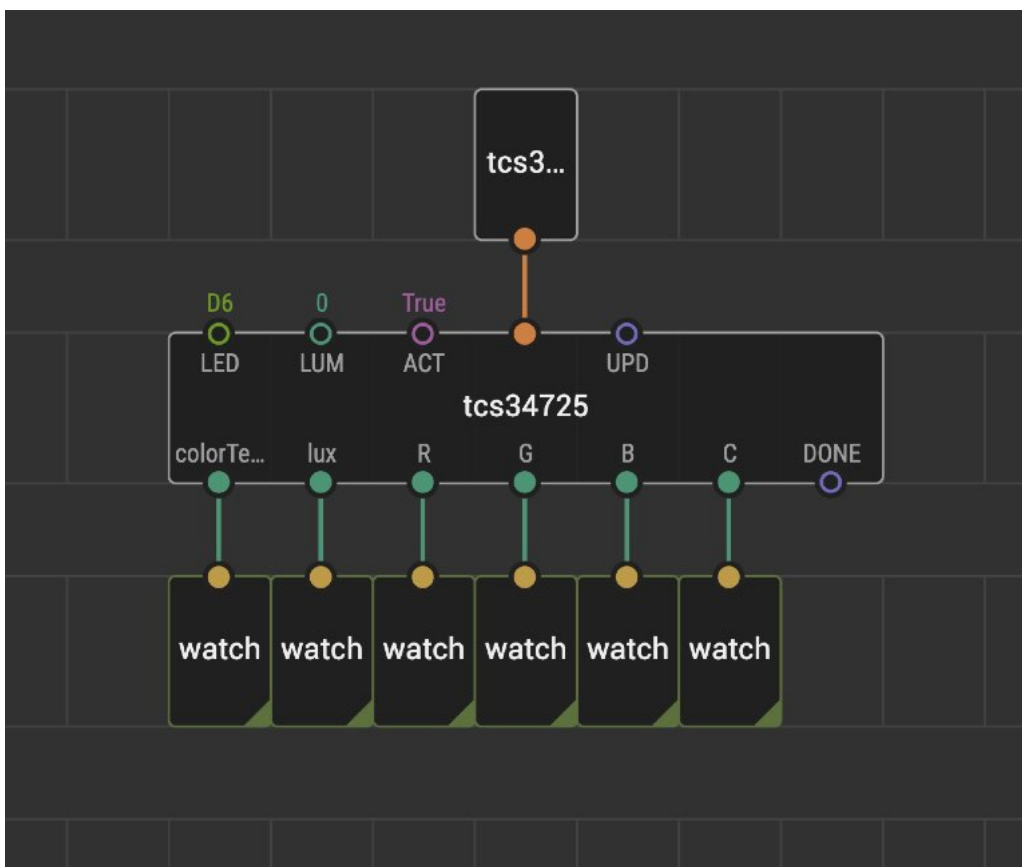
Node to get the data from tcs3475, but the LED is continuously on





## Test patch

1. Connect a TCS3472 or TCS347265 to the Sreedstudio Grove Beginner board - either directly or through a connected breadboard.
2. Load the example patch (**example-tcs3475**) from the **antoniorrg/tcs34725** XOD library
3. Assign the correct port number for the LED, and set the IC2 address for the device. This should be 0x29.
4. Set the LUM input to 1, to turn on the white LEDs, if you wish to examine reflective objects
5. Place coloured objects adjacent to the sensor and read the values for colour temperature, total luminance (lux), red, (615nm) green (525nm), blue (465nm) and clear values.
6. Display values with **watch** nodes, feed to a text or graphical display or use to drive an RGB LED or similar indicator.





# VL6180X Laser Range Finder

## Description

The VL6180X is a Time-of-Flight (ToF) laser-ranging sensor developed by STMicroelectronics. It combines an infrared (IR) emitter, a photodetector, and a microcontroller on a single chip to measure distance and ambient light levels. The sensor is primarily used for proximity detection, distance measurement, and gesture recognition.

The laser range finder works by emitting a short pulse of infrared light, detecting the reflected light from a target object, and measuring the Time-of-Flight to calculate the distance. The sensor processes the ToF data to ensure accurate measurements and communicates with external devices using the I2C protocol.

The VL6180X device includes:

- 1. IR laser emission:** The VL6180X includes a Vertical Cavity Surface Emitting Laser (VCSEL) as an IR emitter. When the sensor is triggered to perform a distance measurement, the VCSEL emits a short pulse of infrared light.
- 2. Light reflection and detection:** The emitted infrared light travels towards the target object and reflects off its surface. The reflected light then returns to the sensor and is detected by the integrated photodetector. This includes silicon photo avalanche diode (SPAD) detectors, ambient light sensors with a sensitivity < 1 Lux up to 100 kLux.
- 3. Time-of-Flight measurement:** The VL6180X measures the time it takes for the emitted light pulse to travel to the target object and back. This time is called the Time-of-Flight (ToF). Since the speed of light is constant, the sensor can calculate the distance to the target object using the following formula:  

$$\text{Distance} = (\text{Speed of Light} \times \text{Time-of-Flight}) / 2$$
 The division by 2 accounts for the round-trip travel of the light pulse.
- 4. Signal processing:** The microcontroller inside the VL6180X processes the ToF data and performs various calculations to compensate for factors such as temperature, ambient light, and crosstalk between the IR emitter and photodetector. This processing ensures accurate and reliable distance measurements.
- 5. I2C communication:** The VL6180X communicates with external devices, such as microcontrollers, using the I2C communication protocol. The processed distance and ambient light data can be read from the sensor's internal registers via I2C commands.
- 6. Programmable settings:** The VL6180X offers various programmable settings, including adjustable ranging resolution, ranging measurement rate, and ambient light sensing rate. These settings allow users to optimize the sensor's performance for specific applications and operating conditions.

The device is one of a family that can be used for different ranges. The VL6180X can handle about 5mm to 100mm of range distance, up to 150-200mm with good ambient conditions. For longer ranges, other devices, such as the VL53L1X can reach out to 4 metres. The devices are capable of relatively high sample rates of 20-50 Hz.

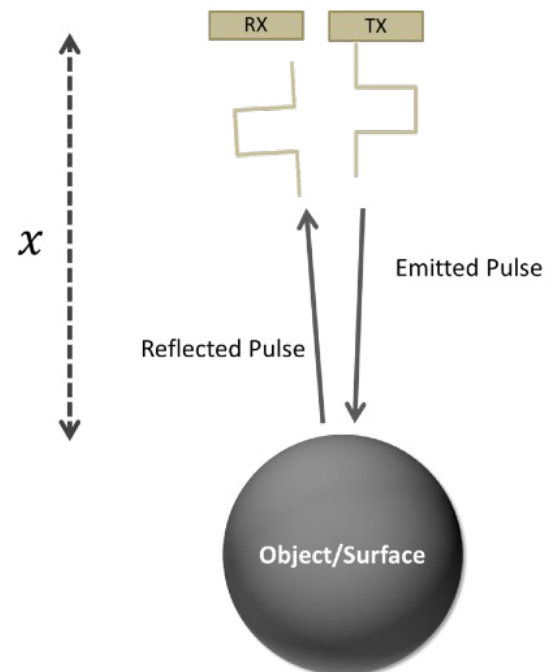
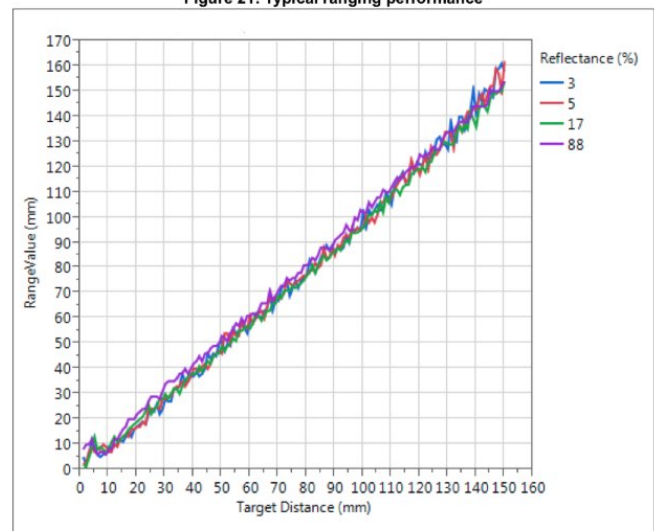
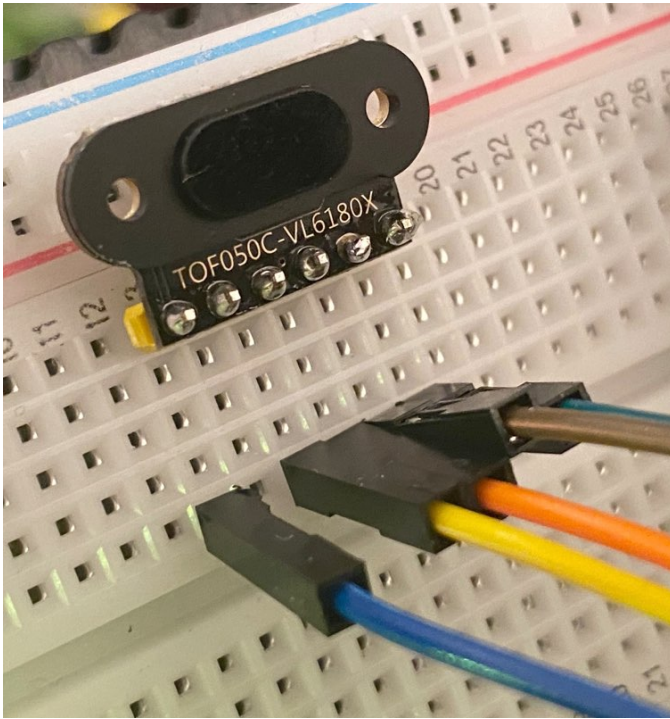


Figure 21. Typical ranging performance

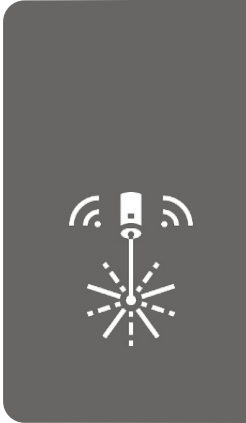




## How to Connect

The sensor board has several plated holes for soldering standard 0.1" spaced pin headers.

- **Vin** - this is the power pin. The chip uses 2.8V, and the board includes a voltage regulator on board that will take 3-5VDC and safely convert it down. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V.
- **GND** - common ground for power and logic
- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line.
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line.
- **GPIO** - this is a pin that is used by the sensor to indicate that data is ready. It's useful for when doing continuous sensing. Note there is no level shifting on this pin, you may not be able to read the 2.8V-logic-level voltage on a 5V microcontroller (we could on an arduino UNO but no promises). Our library doesn't make use of this pin but for advanced users, it's there!
- **XSHUT/SHDN** - the shutdown pin for the sensor. By default it's pulled high. There's a level-shifting diode so you can use 3-5V logic on this pin. When the pin is pulled low, the sensor goes into shutdown mode.



To connect to the Grove board: Connect four I2C pins to corresponding sockets on the central portion of the microcontroller board - either directly, or via the breadboard.

- Connect the GND pin to GND
- Connect the VCC (5v) to 5V
- Connect the SDA pin to SDA
- Connect the SCL pin to SCL

These are the only connections required for normal use on the I2C bus, if you require the device to be shutdown and turned on again (i.e resetting the I2C address), a suitable digital port can be connected to XSHUT/SHDN pin (port D3 is used in the example provided in the [wayland/vl6180x-time-of-flight](#) XOD library).

Selection Table			
Model	TOF050C 	TOF200C 	TOF400C 
Ranging chip	VL6180	VL53L0X	VL53L1X
Measuring distance	50CM (Max)	2M (Max)	4M (Max)
Measurement dead zone	0-2CM	0-3CM	0-4CM
Infrared emission mechanism	850nm	940nm	940nm
FOV	25 °	25 °	27 °
Communication mode	IIC		
Development routines/software	Arduino Demo / STM32 Demo		
Operating Voltage	3.0V-5V (DC)		
Operating current	40mA (Max)		
Operating temperature	-20 ° C-70 ° C		
Storage temperature	-20 ° C-80 ° C		



# Using a VL6180X Laser Range Finder

## XOD library

Load the external library **wayland/vl6180x-time-of-flight**.

The library contains nodes that allow control of the sensor, readout and demonstration patches (right). The example1-test-sensor patch can be used to test the connection to the device. A second patch, example2-change-address, demonstrates how to change the I2C address of the device - the patch will change the address to 30h, from the default 29h. (Careful, you may need to reset the address back to the default if you run this patch).

The VL6180X device is a complex sensor that provides a number of parameters for control, such as gain control, and readouts for status, such as out-of-range distance or high light levels.

## Further information:

How do vertical-cavity surface-emitting lasers (VCSEL) work? - <https://www.sparkfun.com/news/2796> and [https://en.wikipedia.org/wiki/Vertical-cavity\\_surface-emitting\\_laser](https://en.wikipedia.org/wiki/Vertical-cavity_surface-emitting_laser)  
Excellent description of time-of-flight measurements: <https://makersportal.com/blog/2019/4/10/arduino-vl53l1x-time-of-flight-distance-measurement>

VL6180X datasheet: [https://cdn-learn.adafruit.com/assets/assets/000/037/608/original/VL6180X\\_datasheet.pdf](https://cdn-learn.adafruit.com/assets/assets/000/037/608/original/VL6180X_datasheet.pdf)  
Explanation of VL6180X status codes: [https://www.st.com/resource/en/design\\_tip/dt0020-vl6180x-range-status-error-codes-explanation-stmicroelectronics.pdf](https://www.st.com/resource/en/design_tip/dt0020-vl6180x-range-status-error-codes-explanation-stmicroelectronics.pdf)

## wayland/vl6180x-time-of-flight@0.0.5

License: BSD 3-Clause "New" or "Revised" License

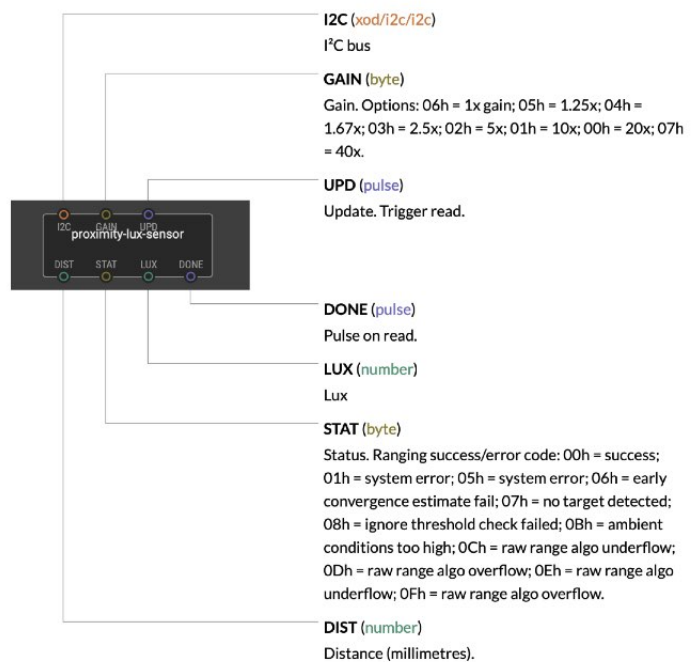
VL6180X proximity and lux sensor. Wraps [https://github.com/adafruit/Adafruit\\_VL6180X](https://github.com/adafruit/Adafruit_VL6180X). Data sheet: [https://cdn-learn.adafruit.com/assets/assets/000/037/608/original/VL6180X\\_datasheet.pdf](https://cdn-learn.adafruit.com/assets/assets/000/037/608/original/VL6180X_datasheet.pdf)

Node	Description
<a href="#">example1-test-sensor</a>	Patch to demonstrate sensor.
<a href="#">example2-change-address</a>	Patch to demonstrate changing I <sup>2</sup> C address of VL6180X.
<a href="#">proximity-lux-sensor</a>	Proximity and ambient light sensor.
<a href="#">init</a>	Initialize VL6180X device.
<a href="#">read-lux</a>	Measure ambient light intensity.
<a href="#">vl6180x-device</a>	Create VL6180X device.
<a href="#">read-range</a>	Read range (mm).
<a href="#">set-address</a>	Set I <sup>2</sup> C address.
<a href="#">translate-status-code-to-text</a>	Translate status code to text string.
<a href="#">get-address</a>	Read the I <sup>2</sup> C address of the VL6180X.

## proximity-lux-sensor

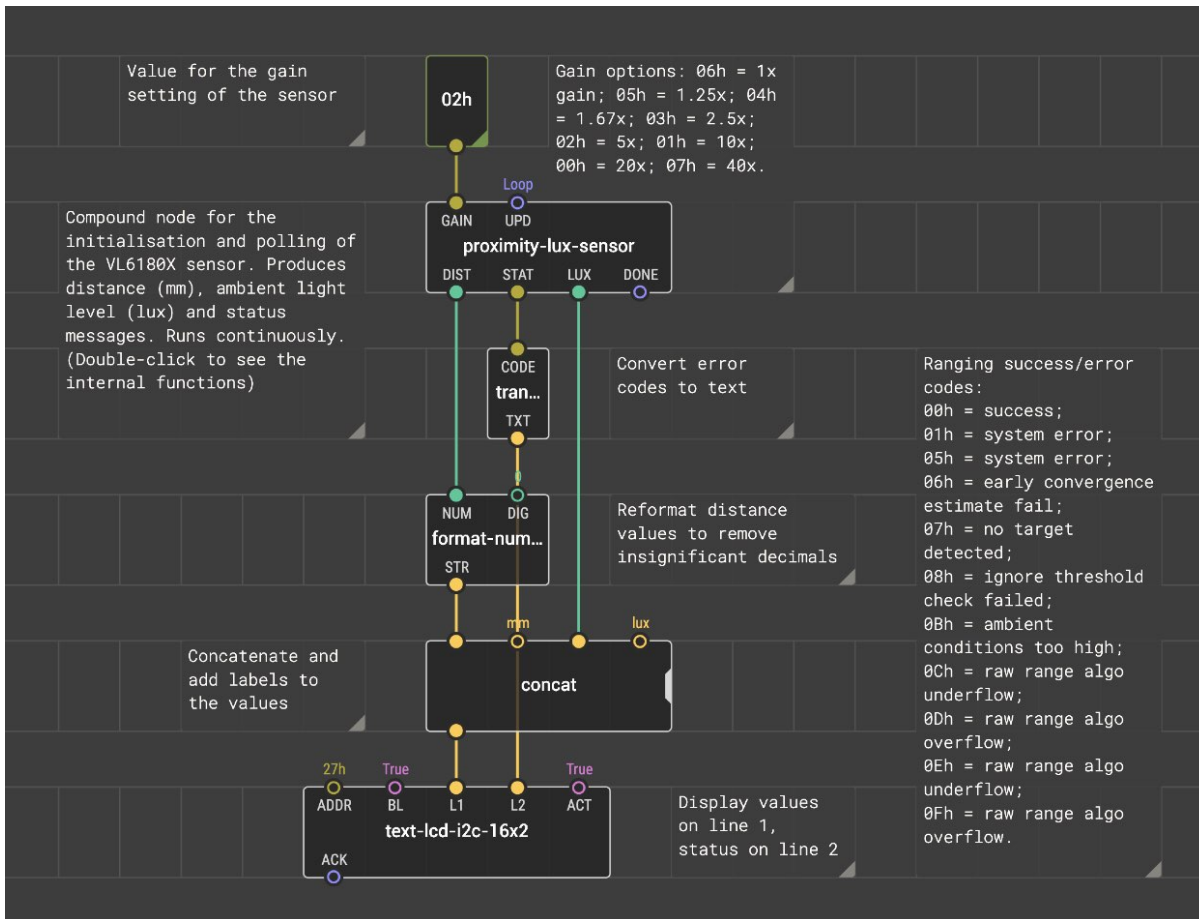
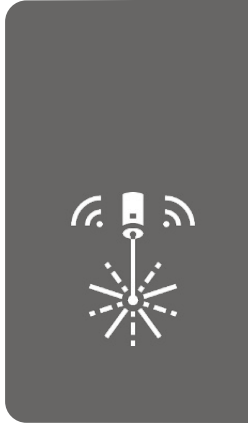
[wayland/vl6180x-time-of-flight/proximity-lux-sensor](#)

Proximity and ambient light sensor.



## Test patch

The patch below takes readings for ambient light and measured distances from the sensor, and displays these on a connected 16x2 LCD display. It allows one to adjust the gain setting on the sensor, and formats the numerical output for display of sensor values and status. It is based on the example patch provided with the **wayland/vl6180x-time-of-flight** library. An example for setting a new I2C address is also provided.



# MCP9808 temperature sensor

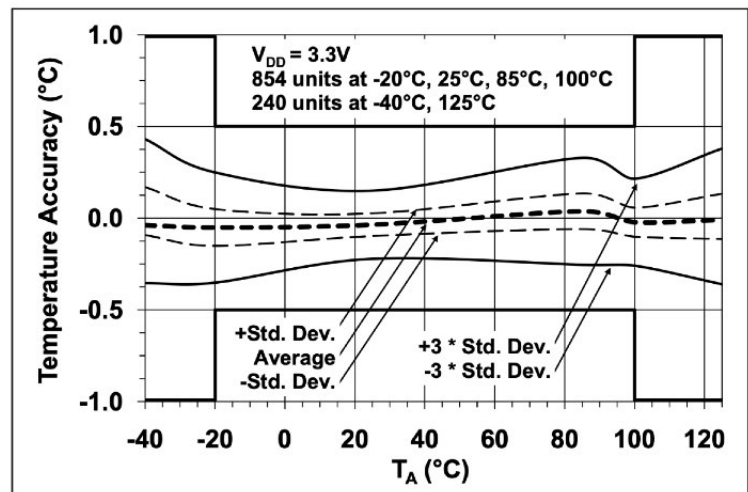
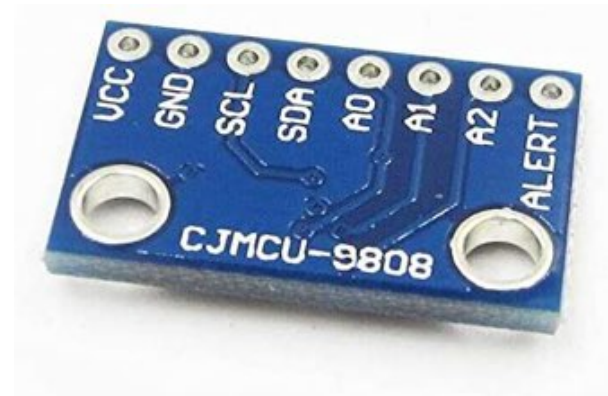
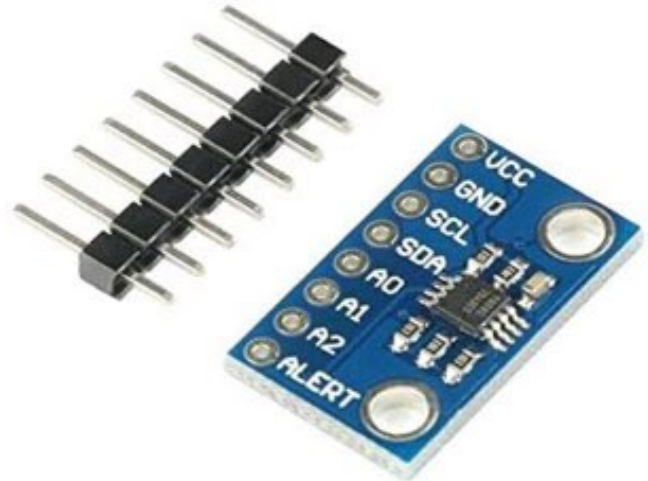
## Description

The Microchip Technology MCP9808 is a digital temperature sensor designed to provide highly accurate temperature measurements over a wide temperature range with low power consumption.

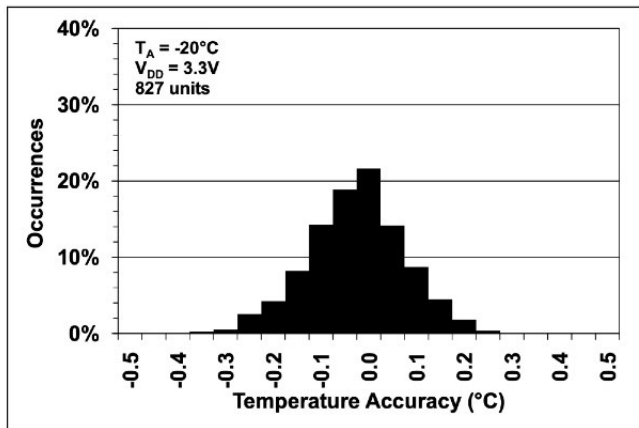
- 1. Temperature sensing:** The core of the MCP9808 is a temperature-sensitive element called a bandgap-based temperature sensor. This sensor generates a voltage that is proportional to the absolute temperature (PTAT), which is then used to determine the temperature reading.
- 2. Analog-to-digital conversion:** The voltage generated by the temperature-sensitive element is converted into a digital value using an integrated analog-to-digital converter (ADC). The MCP9808 features a high-resolution ADC (12-bit or 16-bit, user-selectable), providing accurate and precise temperature measurements.
- 3. On-chip signal processing:** The MCP9808 includes an on-chip signal processing circuit that processes the digital output of the ADC, converts it into a temperature value, and stores it in an internal register. This processing ensures that the temperature data is reliable and ready for use by external devices.
- 4. I2C communication interface:** The MCP9808 communicates with external devices, such as microcontrollers or other host systems, using the I2C communication protocol. It supports standard and fast I2C modes and has a user-selectable 7-bit slave address. The sensor provides various commands for reading the temperature data, configuring the resolution and other settings, and managing alert functions.
- 5. Alert functionality:** The MCP9808 features programmable temperature alert thresholds and outputs. Users can configure the upper and lower temperature limits, as well as the temperature hysteresis value. When the measured temperature crosses the defined limits, the sensor can generate an alert signal to trigger external actions, such as activating a cooling fan or shutting down a system.
- 6. Power supply and low power modes:** The MCP9808 operates with a supply voltage range of 2.7V to 5.5V, making it suitable for a wide range of applications. The sensor also features low power consumption and various power-saving modes, including shutdown and continuous conversion modes, to optimize energy usage.

The MCP9808 temperature sensor is a high accuracy ( $\pm 0.25^{\circ}\text{C}$ ), high precision ( $+0.0625^{\circ}\text{C}$ ) device, with a temperature range of  $-40^{\circ}\text{C}$  to  $+125^{\circ}\text{C}$ . Temperature measurements are based on silicon bandgap properties. There are 3 address pins so you can connect up to 8 devices to a single I2C bus without address collisions. A wide voltage range of 2.7V to 5.5V allows use with a wide variety of 3.3V and 5V logic devices.

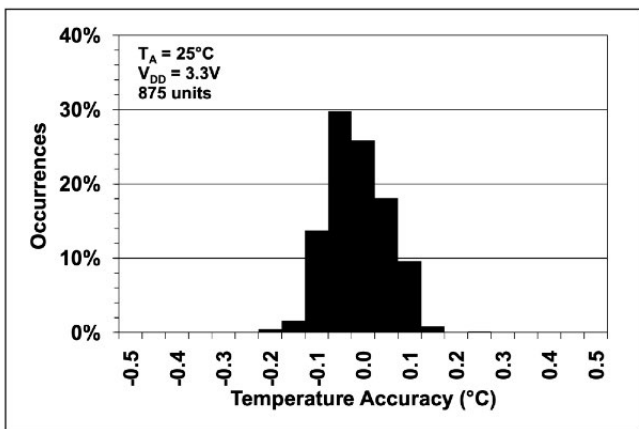
Unlike other digital temperature sensors, like the commonly used DS18B20, it does not come in through-hole package and is most conveniently used on a breakout board PCB. The PCB includes mounting holes, and pull down resistors for the 3 address pins.



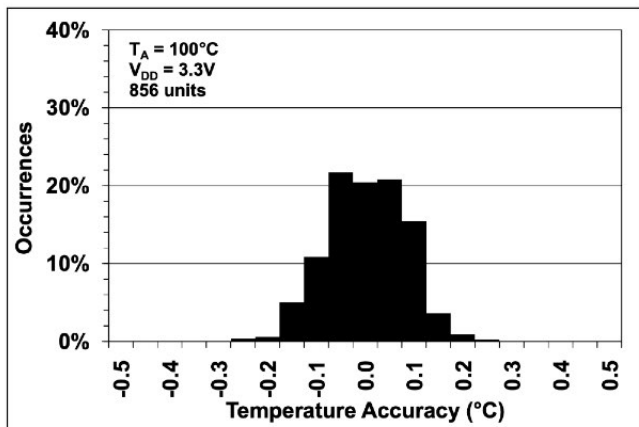
**FIGURE 2-1:** Temperature Accuracy.



**FIGURE 2-2:** Temperature Accuracy Histogram,  $T_A = -20^\circ\text{C}$ .



**FIGURE 2-3:** Temperature Accuracy Histogram,  $T_A = +25^\circ\text{C}$ .



**FIGURE 2-6:** Temperature Accuracy Histogram,  $T_A = +100^\circ\text{C}$ .

## Specifications

Technical specs:

- Sensing Temperature:  $-40^\circ\text{C} \sim 125^\circ\text{C}$
- $0.0625^\circ\text{C}$  resolution
- Accuracy:  $0.25^\circ\text{C}$  typical
- Voltage - Supply:  $2.7\text{V} \sim 5.5\text{V}$
- Operating Current:  $200\ \mu\text{A}$  (typical)
- Operating Temperature:  $-40^\circ\text{C} \sim 125^\circ\text{C}$
- Uses any I2C address from  $0x18$  thru  $0x1F$

## How to Connect

The sensor has 8 pins: VCC, GND, SCL, SDA for I2C connection, A0, A1 and A2 to allow you to change the I2C address of the device (optional), and ALERT pin to allow you to set temperature thresholds (optional).

To connect to the Grove board:

- Connect four wires to pins inside one of the I2C sockets on the central portion of the board (top right)
- Connect left-most wire (GND) to GND pin on module
- Connect the second-left wire (VCC) to VCC pin on module
- Connect the second-right wire (SDA) to SDA pin on module
- Connect the right-most wire (SCL) to SCL pin on module



### 3.5 Address Pins (A0, A1, A2)

These pins are device address input pins.

The address pins correspond to the Least Significant bits (LSBs) of the address bits and the Most Significant bits (MSBs): A6, A5, A4, A3. This is illustrated in [Table 3-2](#).

**TABLE 3-2: MCP9808 ADDRESS BYTE**

Device	Address Code				Slave Address		
	A6	A5	A4	A3	A2	A1	A0
MCP9808	0	0	1	1	x <sup>(1)</sup>	x	x
MCP9808 <sup>(2)</sup>	1	0	0	1	x	x	x

**Note 1:** User-selectable address is shown by 'x'. A2, A1 and A0 must match the corresponding device pin configuration.

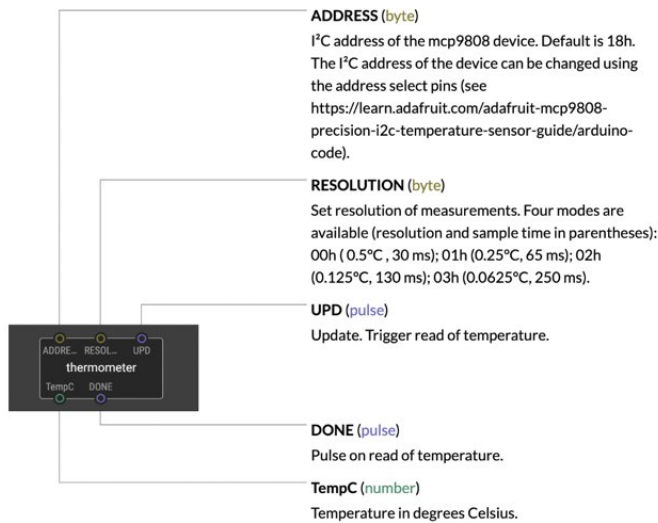
**2:** Contact factory for this address code.



# Using the MCP9808 temperature sensor

## XOD library

Matt Wayland has created the **wayland/mcp9808-thermometer** library to use this sensor in XOD. Use the thermometer node from this library with the default address of 18h. The TempC pin will read out the temperature in degrees Celsius.



## Further information:

Tutorial available here: <https://learn.adafruit.com/adafruit-mcp9808-precision-i2c-temperature-sensor-guide>  
Extreme precision measurements: <https://www.analog.com/en/technical-articles/silicon-temperature-sensing-with-precision.html>

## wayland/mcp9808-thermometer@0.0.2

License: BSD 3-Clause

Precision I<sup>2</sup>C temperature sensor. Arduino library for the MCP9808 sensor (<https://learn.adafruit.com/adafruit-mcp9808-precision-i2c-temperature-sensor-guide>). Wraps [https://github.com/adafruit/Adafruit\\_MCP9808\\_Library](https://github.com/adafruit/Adafruit_MCP9808_Library). Device datasheet: <https://cdn-shop.adafruit.com/datasheets/MCP9808.pdf>

Node	Description
<a href="#">celsius-to-fahrenheit</a>	Convert from Celsius to Fahrenheit.
<a href="#">get-resolution</a>	Get resolution mode of device. MCP9808 has four resolution modes (resolution and sample time in parentheses): 00h (0.5°C, 30 ms); 01h (0.25°C, 65 ms); 02h (0.125°C, 130 ms); 03h (0.0625°C, 250 ms).
<a href="#">mcp9808-device</a>	Create mcp9808 device.
<a href="#">read-temperature</a>	Read temperature in degrees Celsius. Temperature can be converted to Fahrenheit scale using celsius-to-fahrenheit node.
<a href="#">set-resolution</a>	Set resolution mode of MCP9808 device. Four modes are available (resolution and sample time in parentheses): 00h (0.5°C, 30 ms); 01h (0.25°C, 65 ms); 02h (0.125°C, 130 ms); 03h (0.0625°C, 250 ms).
<a href="#">shutdown</a>	Shutdown MCP9808 device. Stops temperature sampling and reduces power consumption of MCP9808 device to ~0.1 microampere.
<a href="#">wake</a>	Wake up MCP9808 and start temperature sampling. Power consumption when sampling is ~200 microampere.
<a href="#">example-change-resolution</a>	Demonstration of how to change the resolution mode of the MCP9808. Here the resolution mode is changed from 03h (0.0625°C) to 01h (0.5°C). Run this patch in the debugger.
<a href="#">example-output-lcd</a>	Patch to demonstrate sending temperature reading to a text LCD.
<a href="#">example-max-sample-rate</a>	Patch to demonstrate how to maximize sampling rate for a given resolution, by not switching off MCP9808 between reads. Run this patch in debugger.
<a href="#">thermometer</a>	Combines low level nodes to create a simple to use thermometer. Outputs temperature in °C (can be converted to °F using celsius-to-fahrenheit node). To conserve energy the MCP9808 is put into sleep mode between temperature readings. You can maximize sampling rate by not switching off MCP9808 between reads (see example-max-sample-rate).
<a href="#">example-test-thermometer</a>	Patch to test thermometer. Run this patch in debugger.

## Test patch

The `wayland/mcp9808-thermometer` library includes a test patch called `example-test-thermometer`.



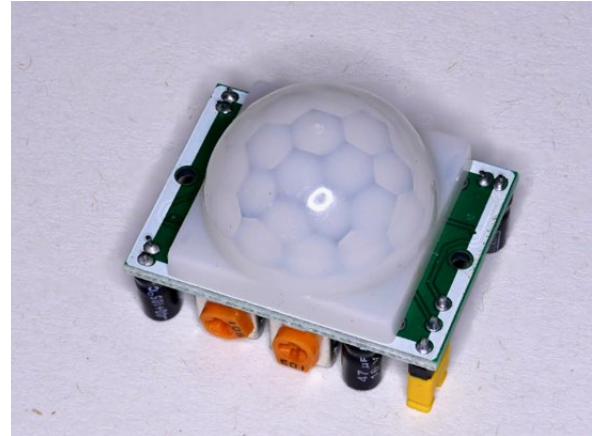


# Motion Sensor (HC-SR501)

## Description

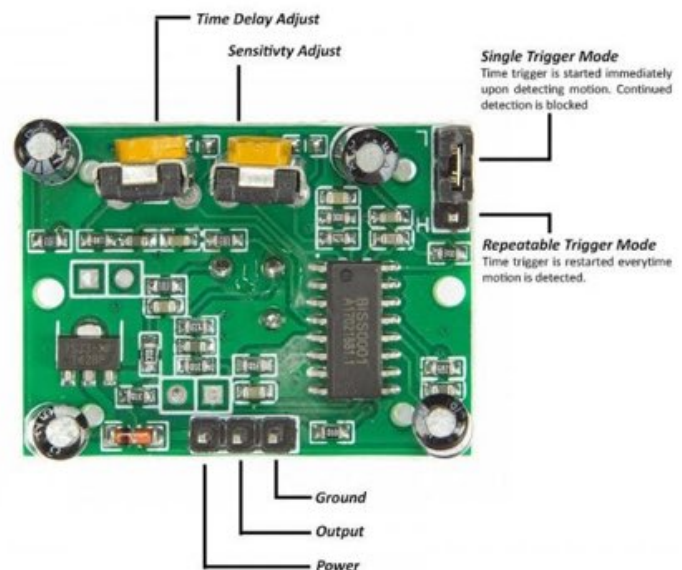
The HC-SR501 is a passive infrared (PIR) motion sensor module designed to detect the presence of humans or animals by sensing the infrared radiation emitted by their bodies.

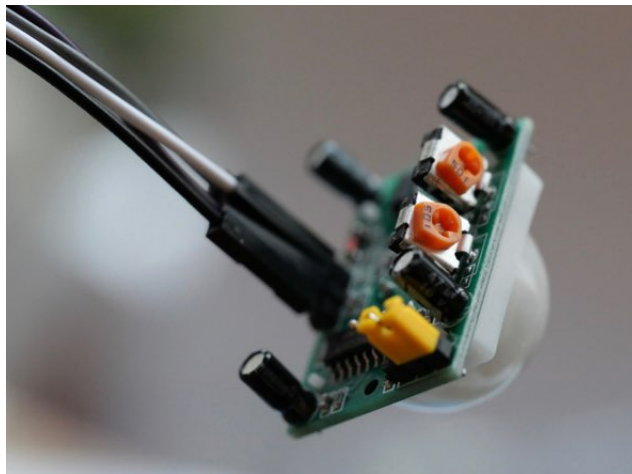
- Infrared radiation sensing:** The core of the HC-SR501 is a pyroelectric infrared sensor, also known as a PIR sensor. This sensor is sensitive to the changes in infrared radiation that occur when a person or an animal moves within its detection range. The surface temperature of the human body is 36-27 ° C, and most of its radiant energy is concentrated in the wavelength range of 8-12 um. The sensor consists of two side-by-side sensing elements, each sensitive to infrared radiation, but with opposite polarities. When a moving object crosses the sensor's field of view, it generates a differential voltage across the two sensing elements.
- Fresnel lens:** The HC-SR501 module includes a white plastic Fresnel lens, which focuses the infrared radiation from the surrounding environment onto the PIR sensor. The lens also increases the detection range and divides the field of view into multiple zones, enhancing the sensor's ability to detect motion more effectively.
- Signal processing:** The differential voltage generated by the PIR sensor is very small and needs to be amplified before further processing. The HC-SR501 module includes an operational amplifier (op-amp) and a comparator to amplify and process the sensor's output. The op-amp amplifies the differential voltage, while the comparator compares the amplified signal with a reference voltage to determine if motion has been detected.
- Adjustable settings:** The HC-SR501 module provides adjustable settings for sensitivity and time delay. The sensitivity control adjusts the detection range of the sensor, while the time delay control determines how long the output signal stays active after motion is detected. These settings can be fine-tuned using the potentiometers on the module to suit different applications and environments.
- Output signal:** When motion is detected, the comparator output goes high (logic level 1), which can be connected to a microcontroller or other external devices to trigger actions such as turning on a light, sounding an alarm, or activating a motor. When no motion is detected, the output stays low (logic level 0).



## Adjustments

The sensor properties are adjustable, and a switch and adjustable potentiometer are provided for this. It has two modes Repeatably Trigger (H) and Single Trigger (Non-Repeatably)(L). H mode is the default. The mode can be set using the jumper pins on the bottom side of the board. To switch to L mode, pull the jumper shunt off the bottom two pins, and place it on the top two pins. In Repeatably (H) mode, the sensor will send a high voltage when movement is detected within range, and will stay high for a set amount of time (T) before returning to low. The output will be the same whether the person is still in range or not. This sensor will reset the timer (which would otherwise turn the output off) each time motion is detected; this would be applicable, for example, for room occupancy lighting control where you don't want the lights to blink off while the unit resets.





In Non-Repeatable (L) mode, the sensor will go high when someone enters the range and will stay high until they leave the range.

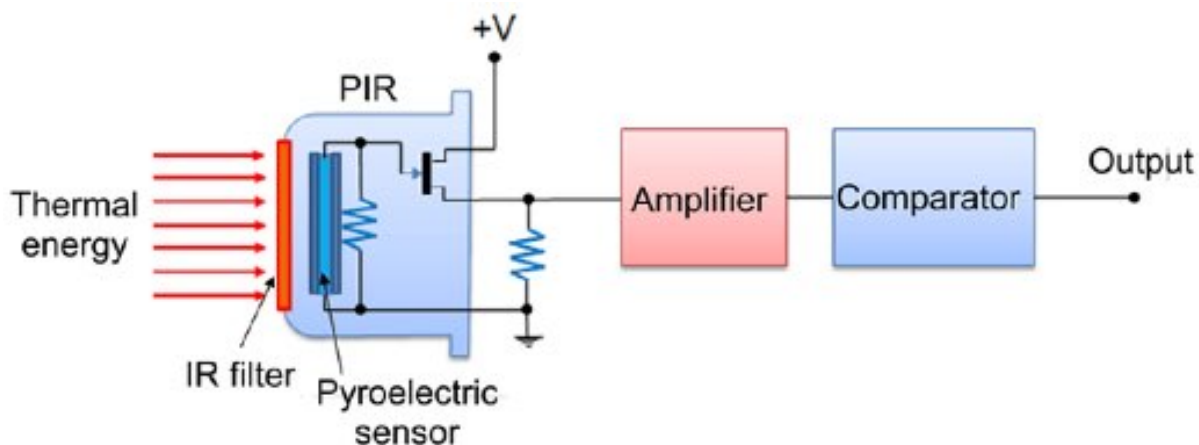
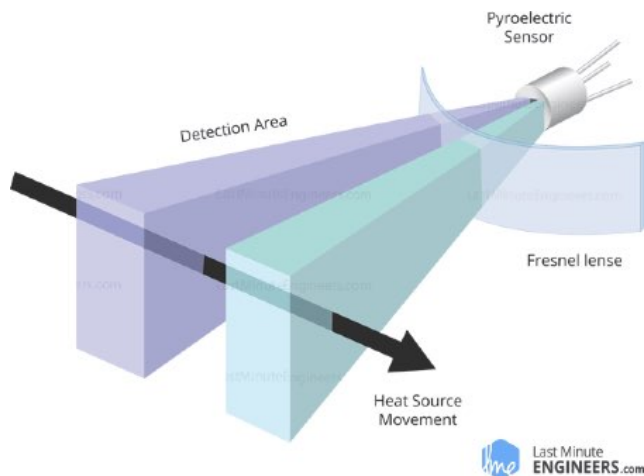
The time parameter (T - for H mode) and sensitivity (S) can be adjusted using the orange potentiometer pins on the underside of the board, using a small flat-bladed screwdriver. For proper calibration, there should not be any movement in front of the PIR sensor for up to 15 seconds (to allow self-calibration). After this period, the sensor has a snapshot of its viewing area and it can detect movements. When the PIR sensor detects a movement, the output will be HIGH, otherwise, it will be LOW.

### How to Connect

The sensor has three pins - VCC, OUT and GND (labels may be below the white plastic lens). To connect to the Grove Beginner Kit board:

Choose an unused digital port (We are using either D2 or D10 in the examples below)

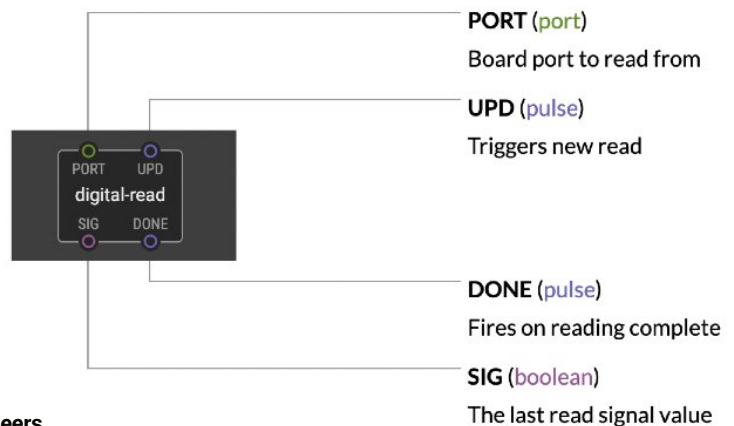
- Connect three pins to the connector in the central portion of the board using Dupont jumper cables (this can be either a direct connection, or via the breadboard)
- Follow the pin identifications shown directly above. (pin labels on the PIR sensor may be underneath the white plastic lens - you can lift the lens to check pin names and see the pyroelectric sensor)
- Connect the Ground pin on the PIR module to a GND socket on the microcontroller connector
- Connect the Power pin on the PIR module to a 5V socket on the microcontroller connector
- Connect the Output pin on the PIR module to the chosen digital port socket on the microcontroller connector



# Using the HC-SR501 motion sensor

## XOD library

The output from the sensor is a simple digital on/off signal. Use the the digital-read node from the **xod/gpio** library. The SIG output pin will read 'true' if movement has been detected and 'false' if not.

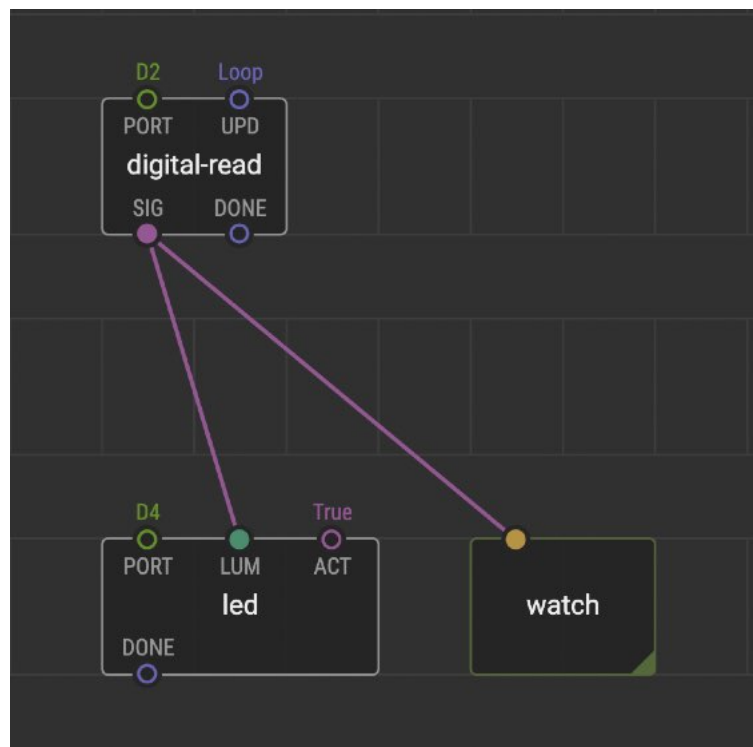


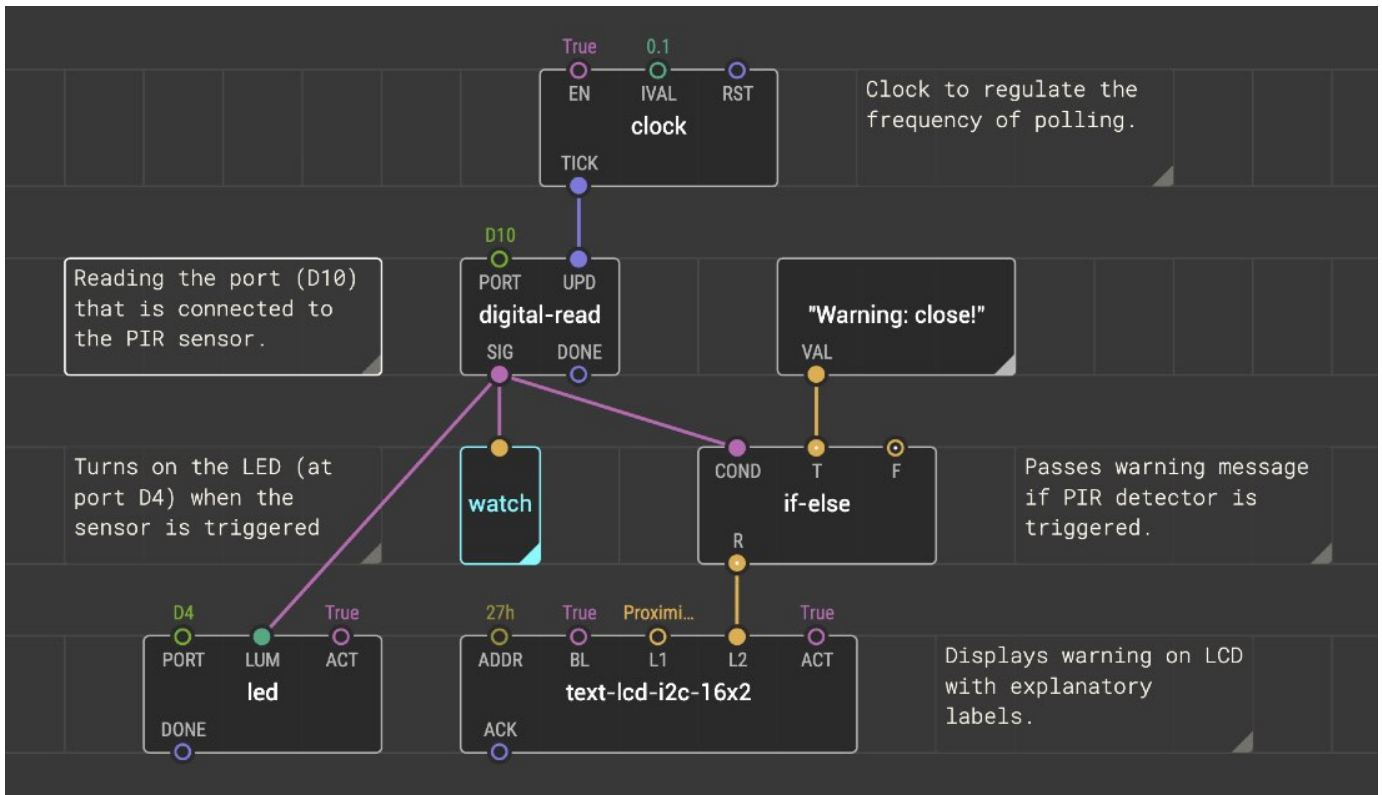
## Further information:

How does a PIR motion sensor work? See: <https://lastminuteengineers.com/pir-sensor-arduino-tutorial/>  
Working principle: <https://robu.in/pir-sensor-working-principle/>

## Specifications

- Wide range of input voltages varying from 4.5V to 20V (+5V recommended)
- Output voltage is High/Low (3.3V TTL)
- Can distinguish between object movement and human movement
- Has to operating modes - Repeatabl(H) and Non- Repeatabl(H)
- Cover distance of about 120° and 7 meters
- Low power consumption of 65mA
- Operating temperature from -20° to +80° Celsius
- **Delay time:** 5-200S (can be adjusted, default 5s +-3%)
- **Blockade time:** 2.5 S (default)
- **Mode adjustment:** yellow jumper pin on underside of the board, pull up yellow jumper shunt and place over bottom to pins for H mode and top two pins for L mode, H is default
- **Sensitivity adjustment:** orange potentiometer screw next to mode pins, switch between 3-7 meters, clockwise to high anti-clockwise to low
- **Time adjustment:** orange potentiometer screw further away from mode pins, switch between 3 sec - 5 min, clockwise to long anti-clockwise to short





## Test patches

### Test patch 1 (left)

Set port to D2 (or whichever port you are using) and UPD to 'Continuously'. Connect SIG to a watch node, or to an led node to get a visual output. Upload and debug.

### Test patch 2 (above)

Here the PIR detector is connected to port D10. The status of the sensor (digital-read) is polled every 0.1 seconds, driven by pulses from the clock node. The output of the device is sent to a watch node and the on-board LED (led node). The PIR detector status is also sent to an if-else node. If the detector is triggered, the LED will light, and the true condition results in sending the "Warning: close" message to the LCD display via the text-lcd-i2c-16x2 node. A "Proximity:" label is displayed on the first line of the display.





# Radar Proximity Sensor (RCWL-0516)

## Description

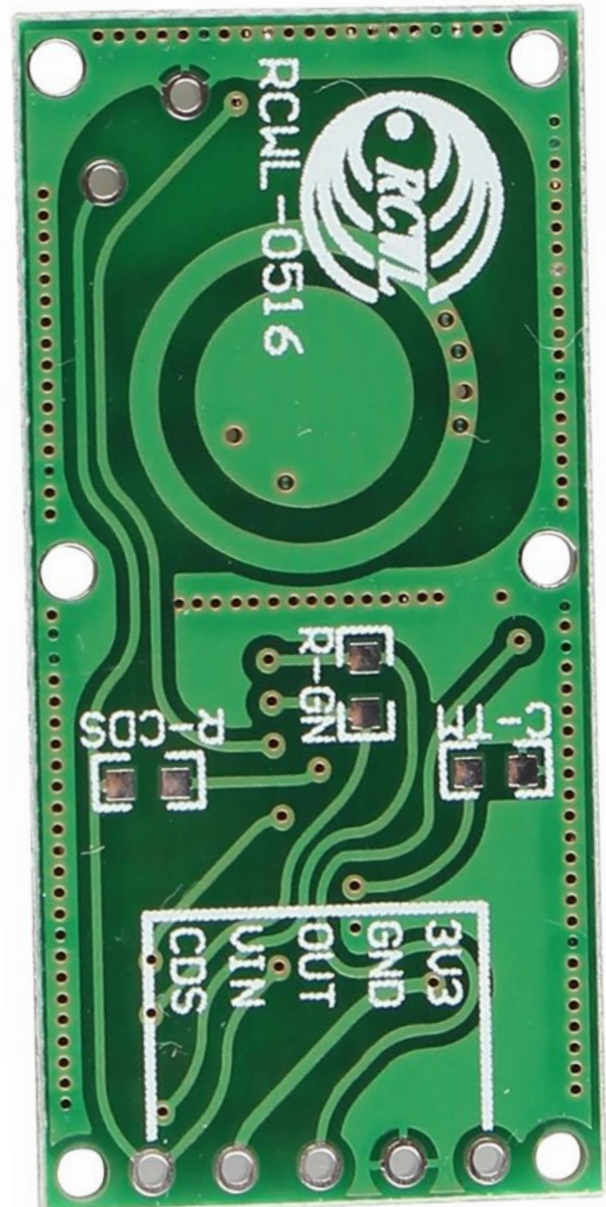
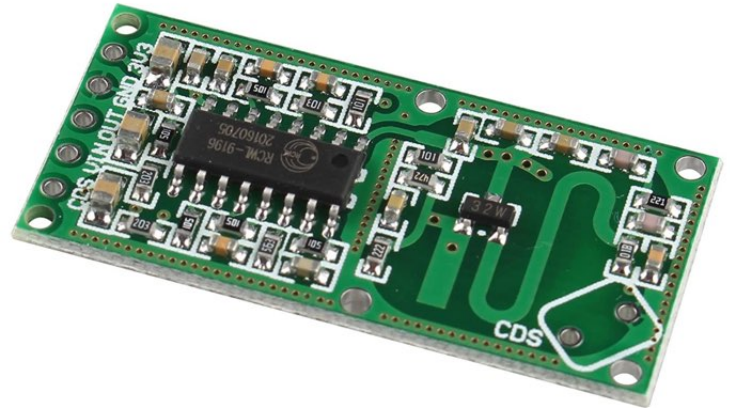
The RCWL-0516 is a radar-based proximity sensor module that detects the presence of objects, humans, or animals within its sensing range. It uses microwave frequency signals to detect motion.

- Microwave signal emission:** The RCWL-0516 module includes a microwave generator that emits continuous-wave microwave signals at a specific frequency, usually around 3.2 GHz. These signals propagate outward from the module's antenna.
- Signal reflection and reception:** When an object, person, or animal moves within the sensing range of the module, the emitted microwave signals reflect off the object and return to the module. The module's antenna receives the reflected signals.
- Doppler effect:** As the object moves, the frequency of the reflected signals changes due to the Doppler effect, a phenomenon discovered in 1842 by the Austrian physicist Christian Doppler. The Doppler Effect describes a change in frequency observed by a stationary observer when the source of the frequency is moving.. The Doppler effect causes the frequency of the reflected signals to increase when the object is moving towards the sensor and decrease when moving away from it. This change in frequency is directly related to the speed of the object.
- Signal processing:** The RCWL-0516 module processes the received signals to extract the Doppler frequency shift. The module includes a mixer that combines the emitted and received signals, producing an output signal with a frequency equal to the difference between the emitted and received signals (the Doppler shift). This output signal is then filtered and amplified to obtain a clear and usable signal.
- Motion detection:** The processed Doppler shift signal is analyzed to determine if motion has occurred. If the signal's amplitude exceeds a predetermined threshold, the module concludes that motion has been detected and triggers an output signal.
- Output signal:** The RCWL-0516 module provides a digital output signal that goes high (logic level 1) when motion is detected and stays low (logic level 0) when no motion is detected. This output signal can be connected to a microcontroller or other external devices to trigger actions, such as turning on a light, sounding an alarm, or activating a motor.

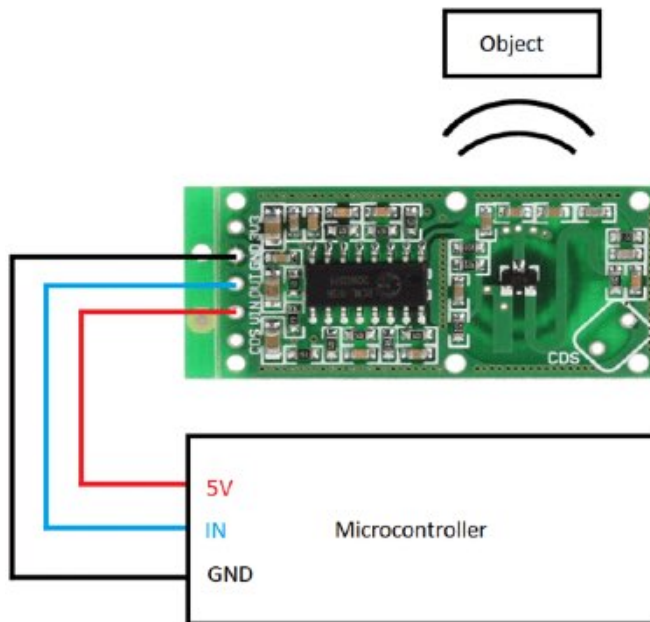
Device operates at ~3.2GHz/20 - 30mW. The device has a sensitivity range of ~7 meters. When triggered its trigger output pin will switch from (LOW) 0V to high (3.3V) for ~ 2 to 3 seconds before returning to its idle LOW state. The device does not measure the distance to the detected object or its velocity, just its presence.

Component side should face detection area for maximum sensitivity. Do not place metal objects within 1cm of antenna. Trigger out - HIGH (3.3V) motion / LOW no motion. Note that the 3V3 pin is a 3.3-volt output, not a power supply input. The device has an integrated 3.3 volt voltage regulator which can provide up to 100 mA of current for powering external logic circuitry.

The VIN pin is the positive power supply input, accepting any voltage from 4-volts to 28-volts. The RCWL-0516 does not consume very much current and can easily be powered by the 5-volt output from an Arduino or a Raspberry Pi. The microwave antennas are integrated onto the small printed circuit board, making it a self-contained unit.







The RCWL-0516 also support an optional cadmium disulphide light-dependent resistor (LDR) to allow the device to operate only in darkness. This can be very useful in light control applications. The light dependent resistor (LDR) can be attached to the "CDS" pins. The LDR can actually be hooked up two ways: (i) Using the two CDS pads on the top of the sensor printed circuit board. (ii) By connecting one end to the CDS pin on the main terminal section and the other end of the LDR to ground. The light sensor will disable the device when it detects ambient light. You may also use the CDS pin as an Enable control for the module.

Pin 9 is pulled up (=output enable) by a 1M resistor. Attaching the optional CDS LDR will pull pin 9 down (=output disable) when it is light (i.e. the LDR's resistance drops below ~269k assuming no resistor R-CDS installed). R-CDS allows you to add a resistance in parallel with the onboard 1M pull-up resistor to adjust the light level at which pin 9 is pulled <0.7V.

The RCWL-0516 radar proximity sensor offers advantages over other types of motion sensors, such as the ability to see through non-metallic materials and immunity to environmental factors like temperature, humidity, and ambient light.

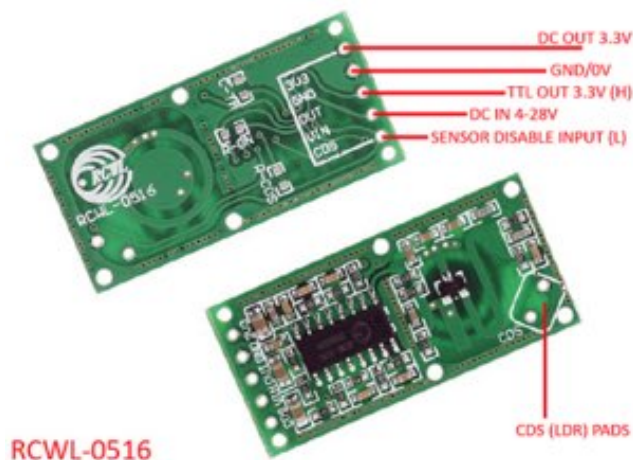
## How to Connect

RCWL-0516 Pinout:

- VIN – 4V - 28V DC power supply input
- CDS – Sensor disable input (low = disable) (For LDR sensors)
- GND – Ground
- 3volt– DC output (100 mA maximum)
- OUTPUT – HIGH /LOW(3.3 V) (according to the motion detection)

To connect to the Grove board:

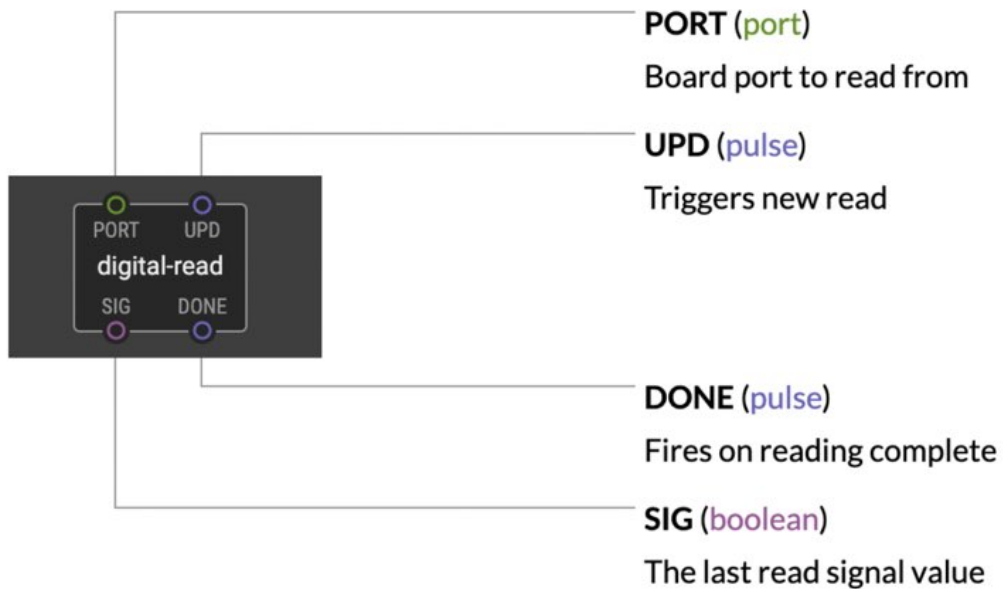
1. Solder pins to the RCWL-0516 board
2. Connect left-most wire (GND) to GND pin on module
3. Connect the second-left wire (VCC) to VIN pin on module
4. Connect the right-most wire (D2) to OUT pin on module



# Using the RCWL-0516 proximity sensor

## XOD library

The sensor provides a simple output signal. Use the the **digital-read** node from the **xod/gpio** library. The SIG output pin will read 'true' if movement has been detected and 'false' if not.



## Further information:

Basic circuit for use of the RCWL-0516: <https://maker.pro/arduino/tutorial/arduino-motion-detector-using-a-microwave-proximity-sensor>

Experiments with the RCWL-0516: <https://dronebotworkshop.com/rcwl-0516-experiments/>

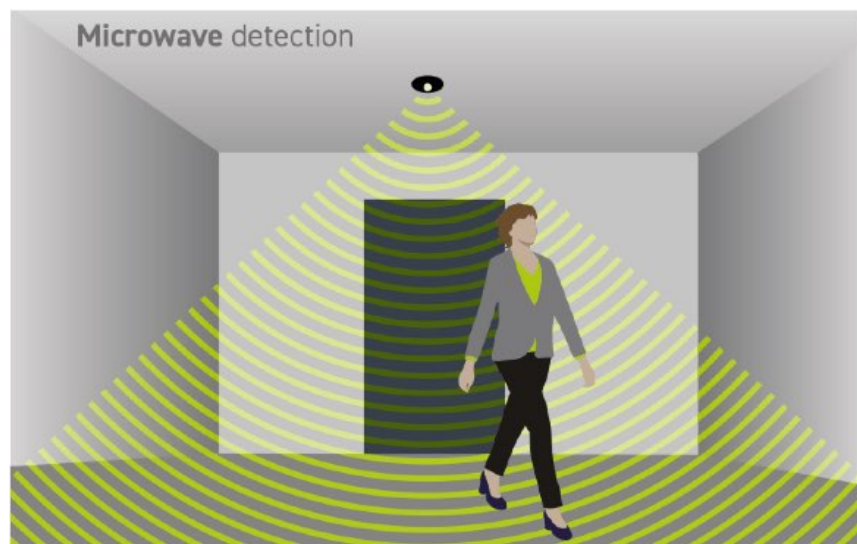
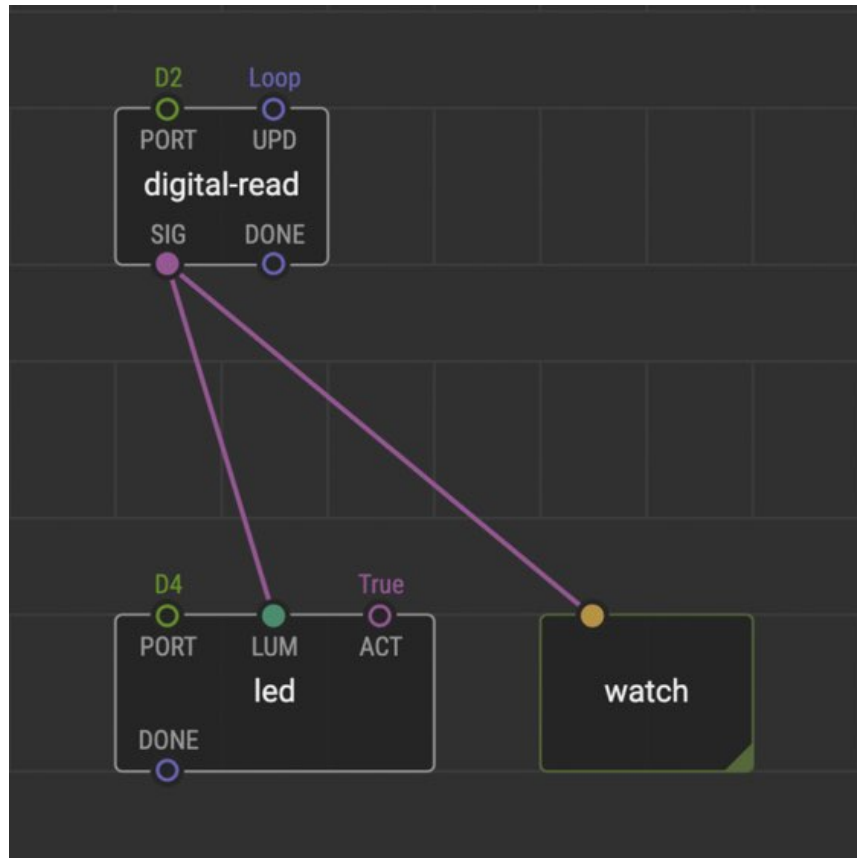
RCWL-0516 information: <https://github.com/jdesbonnet/RCWL-0516>

Component info: <https://components101.com/sensors/rcwl0516-microwave-diatance-sensor-module-datasheet-pinout-features-working>

Electroschematics info: <https://www.electroschematics.com/get-started-microwave-radar-motion-sensor/>

## Test patch

Set port to D2 (or whichever port you are using) and UPD to 'Continuously'. Connect SIG to a watch node, or to an led node to get a visual output. Upload and debug.

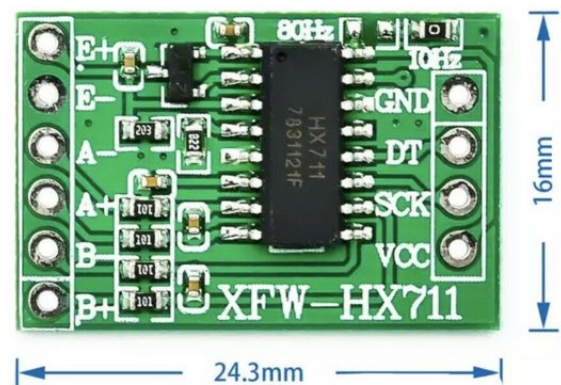
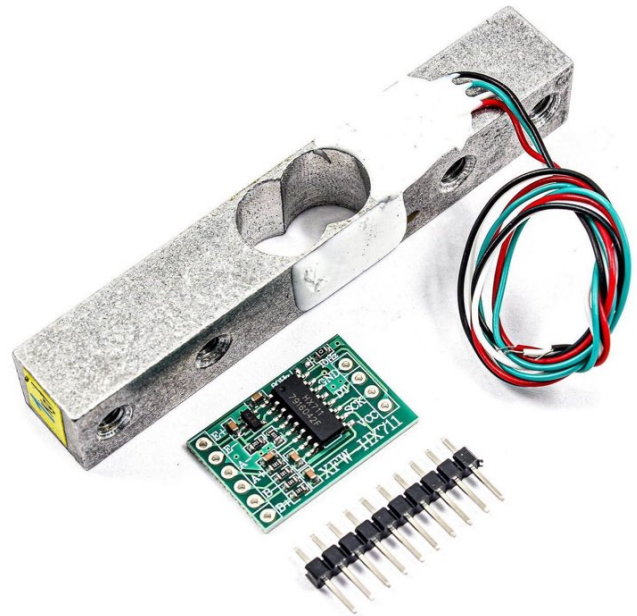


# Weight sensor (HX711)

## Description

The HX711 is a precision 24-bit analog-to-digital converter (ADC) designed for weight scale applications and other force measurement systems. It is typically used in conjunction with load cells, which are sensors that convert mechanical force into an electrical signal. The HX711 amplifies the small electrical signal from the load cell and converts it into a digital value, enabling accurate weight measurements.

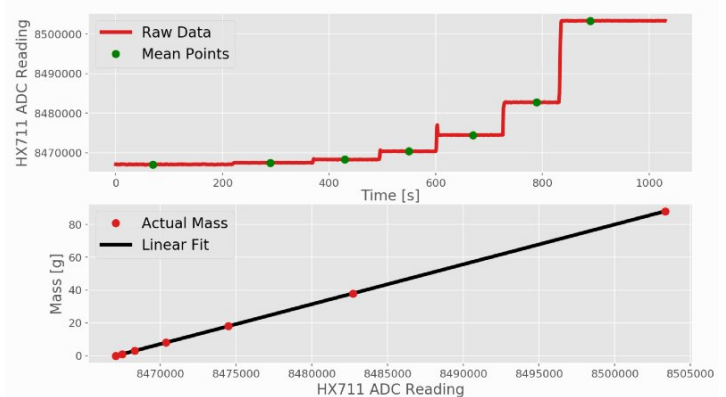
- 1. Load cell sensing:** Load cells are the primary sensing elements used in weight scales. They are typically made from a strain gauge bonded to a metal structure. When a weight is applied to the load cell, the metal structure deforms, causing the strain gauge to change its resistance. This change in resistance is proportional to the applied force or weight.
- 2. Wheatstone bridge:** Load cells are often configured as a Wheatstone bridge, which is a circuit arrangement that allows for accurate measurement of the change in resistance. The Wheatstone bridge converts the change in resistance caused by the applied force into a small differential voltage.
- 3. Signal amplification:** The differential voltage generated by the load cell is very small (in the range of millivolts) and needs to be amplified before further processing. The HX711 includes a programmable gain amplifier (PGA) to amplify the small voltage signal from the load cell. The gain can be set to either 32, 64, or 128, depending on the application requirements and the desired sensitivity.
- 4. Analog-to-digital conversion:** The amplified voltage signal is then converted into a digital value using the integrated 24-bit analog-to-digital converter (ADC) in the HX711. The high-resolution ADC ensures accurate and precise weight measurements.
- 5. Data output and communication:** The HX711 communicates with external devices, such as microcontrollers, using a simple two-wire serial interface. The digital weight data can be read from the HX711 by sending clock pulses to the module and reading the serial data output. The HX711 supports a simple communication protocol that requires no additional components or complex programming.
- 6. Power management:** The HX711 operates with a supply voltage range of 2.6V to 5.5V and includes power management features such as power-down and sleep modes. These features help to reduce power consumption in battery-operated devices and other low-power applications.

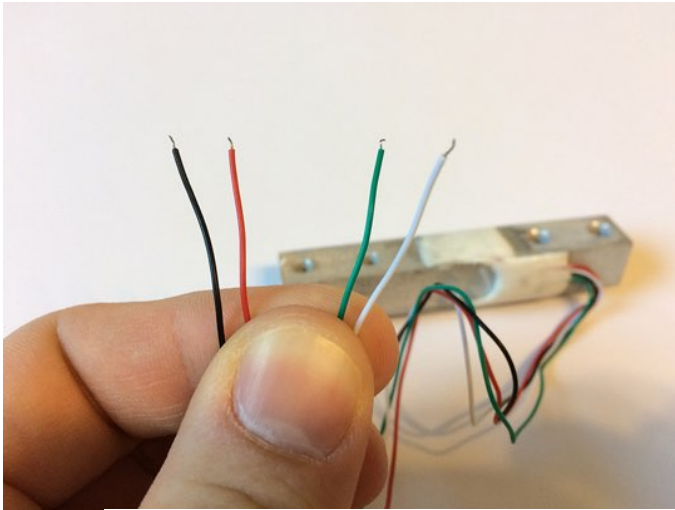


## Load cells

Common straight bar load cells are made from an aluminum alloy and are capable of reading a capacity of 1Kg/2Kg/3Kg/5Kg/10Kg/20Kg. These load cells have four strain gauges that are hooked up in a Wheatstone bridge formation. The load cells feature two M5 sized through-holes for mounting purposes. The load cell needs to be mounted in a way that allows a supported weight to apply a direct lateral force.

A load applied to a strain gauge triggers a change in resistance that can produce an output voltage proportional to the applied load. This relationship between strain and voltage can be used

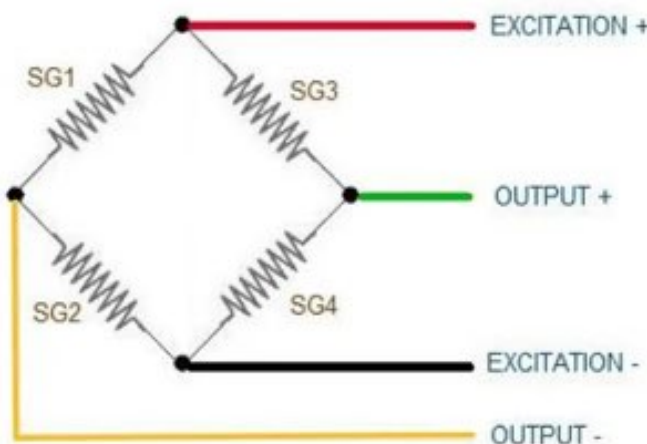
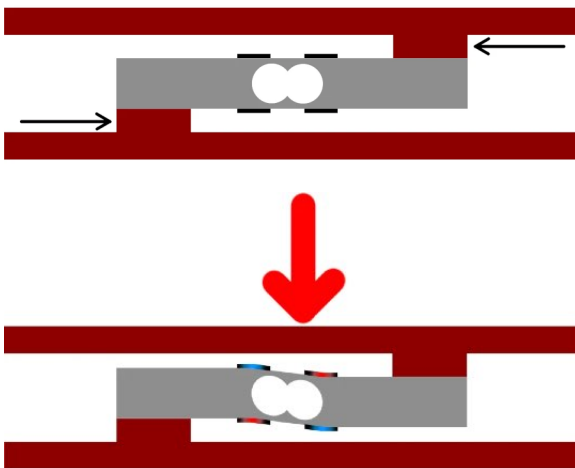




for calibrated weight measurement. Load cells are commonly used due to their linearity, cost effectiveness, and ease of implementation.

The HX711 is a 24-bit analog-to-digital converter translates the small changes in strain from the load cell into 24-bit changes in voltage (Arduino 0-5V). This allows the Arduino to resolve weight (mass) changes down to the range of the load cell (typically 500g, 1kg, 5kg, or more) divided by half the bit depth ( $2^{\text{exp}23}$ ). For a 1kg load cell, this results in mass change detection down to 0.0001g. In practice, however, the analog-to-digital converter and load cell both can have inherent noise (electrical and mechanical) which results in a much lower precision closer to 0.1% of the measurement value.

**Model: YZC-133** (as supplied with the Stage 2 Biomaker kit), Rated Load: 1Kg, Rated Output: 1.0 0.15mV / V, Nonlinear: 0.03% F.S, Hysteresis: 0.03% F.S, Repeatability: 0.03% F.S, Creep (5 minutes): 0.05% F.S, Temperature Effect on Output: 0.003% F.S / C, Temperature Effect on Zero: 0.02% F.S / C, Zero Balance: 0.1000 mV / V, Input Impedance: 1066  $\pm$ 20%  $\Omega$ , Output Impedance: 1000  $\pm$ 10%  $\Omega$ , Insulation Resistance: 2000 M $\Omega$ , Recommended Operating Voltage: 5V, Maximum Operating Voltage: 10V, Material: Aluminum



## How to Connect

1. **Connect the red wire to the E+ and the black wire to the E- output of the HX711 module.** Choose the red and black wire pair to be the power wires of the load cell. E+ and E- are the sensor power outputs of the HX711 module. The polarity doesn't matter. Pick red to be the positive side and black to be the negative side to follow a common convention. Switching red and black will only invert the calibration parameter in the software.
2. **Connect the green wire to the A+ and the white one to the A- inputs of the HX711 module.** A+ and A- are the measurement inputs of the HX711 module. Like with the power wires, the polarity is not important. You just need to recalibrate in the software if you switch them.
3. **Connect the GND of the HX711 module to the Arduino GND and VCC to the Arduino 5V pin.** HX711 also works with 3.3V. If you have some other microcontroller that runs on 3.3V, you can use 3.3V instead of 5V.
4. **Connect the DT and SCK of the HX711 module to any of the Arduino digital IO pins.** In the schematic, use pins 4 and 5, since those are the default pins for the examples of the "HX711\_ADC" library. If you want to use interrupts to update scale data, then you should connect the DT output to an interrupt enabled pin of the Arduino. For Uno/Nano, those are pins 2 and 3.



# Using a component

## XOD library

The external library **gabbapeople/hx711** should be loaded into XOD. This library provides a series of nodes and patches that support the HX711.

## Further information:

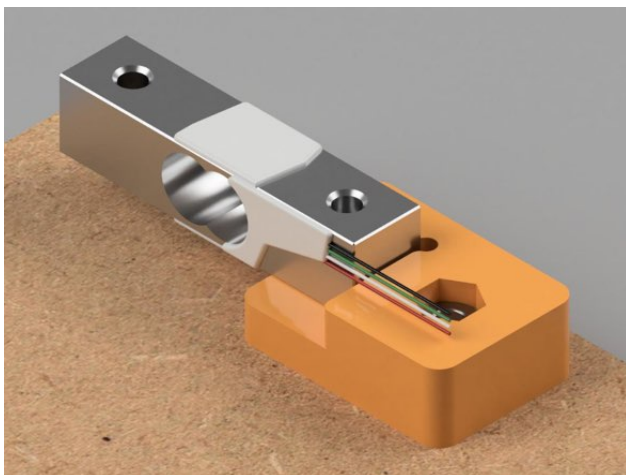
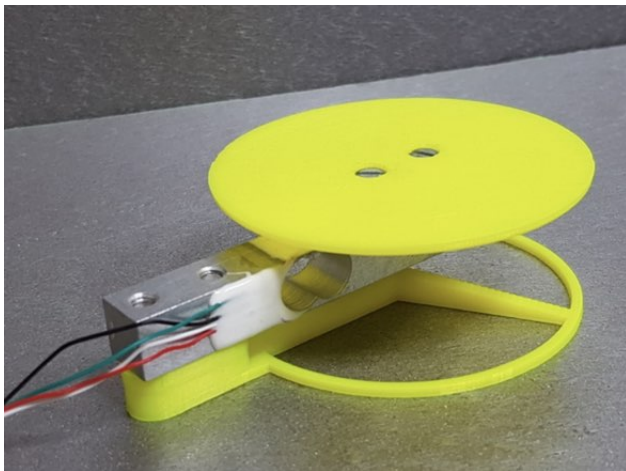
Source of item: <https://www.aliexpress.com/item/1005001418508621.html>

Arduino Weighing Scale with Load Cell and HX711: <https://makersportal.com/blog/2019/5/12/arduino-weighing-scale-with-load-cell-and-hx711>

How load cells work: <https://www.anyload.com/load-cell-force-transducer-how-it-works/>

Getting Started with Load Cells: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>

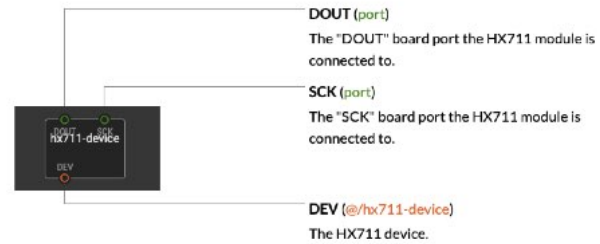
There are a number of designs for 3D printed stands that can be adapted for the provided load cells. 3D printed mounts to support load cells: <https://www.thingiverse.com/thing:3129439> and <https://www.prusaprinters.org/prints/32094-load-cell-mount>



## hx711-device

[gabbapeople/hx711/hx711-device](#)

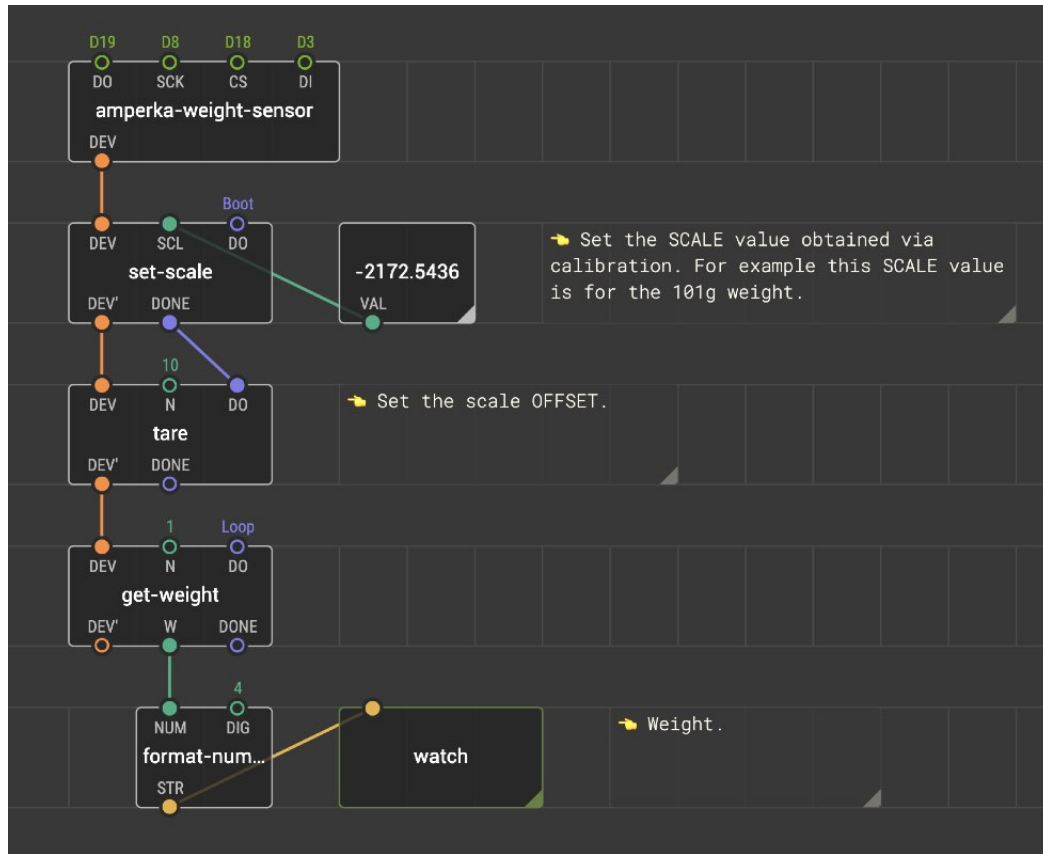
The main device node for the HX711 modules.



## gabbapeople/hx711@0.0.3

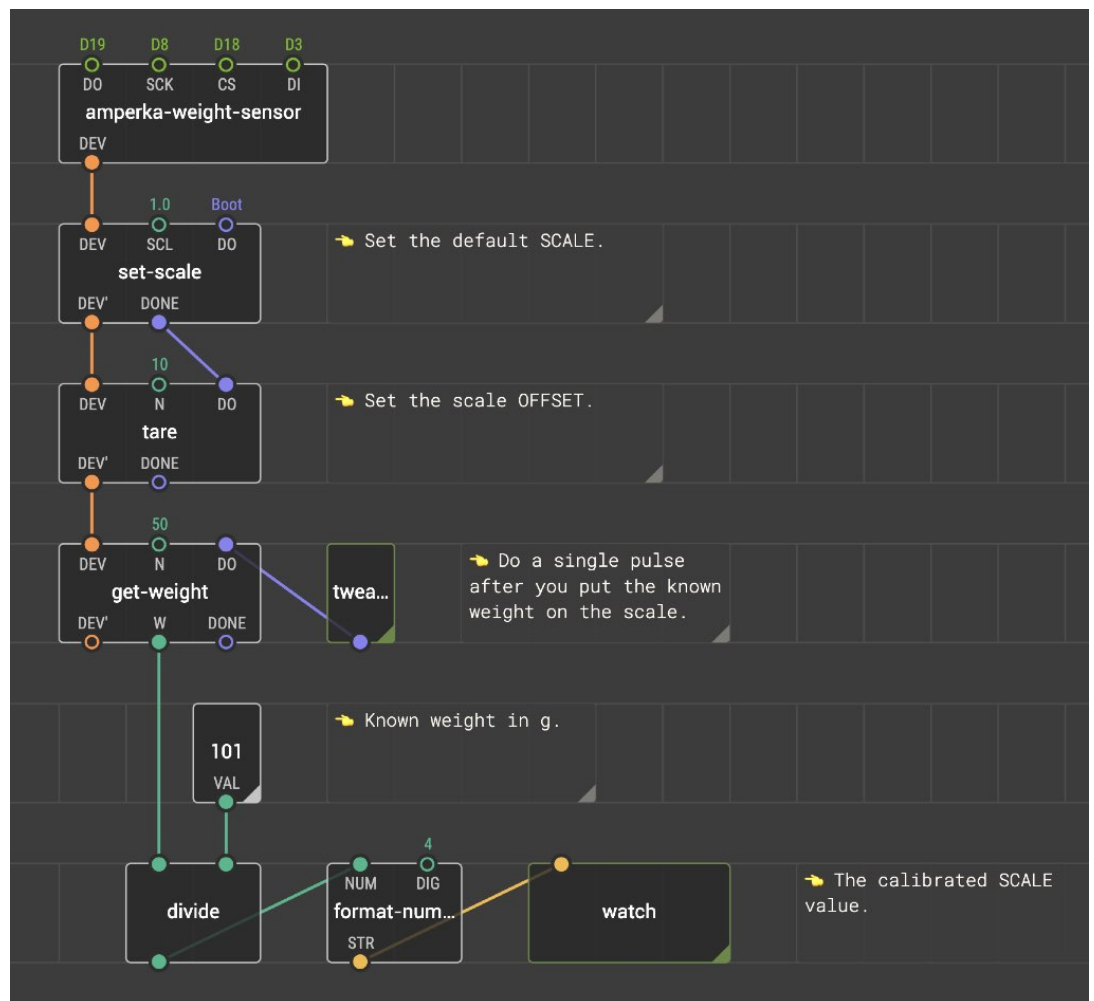
A library to interface HX711 24-Bit Analog-to-Digital Converter (ADC) for reading load cells / weight scales.

Node	Description
<a href="#">hx711-device</a>	The main device node for the HX711 modules.
<a href="#">tare</a>	Sets the OFFSET value for tare weight;
<a href="#">set-scale</a>	Sets the SCALE value. This value is used to convert the raw data to "human readable" measure units. 1.0 by default.
<a href="#">amperka-weight-sensor</a>	Drives the Weight sensor based on the HX711 24-Bit Analog-to-Digital Converter (ADC) by Amperka.
<a href="#">example-calibrate-scale</a>	The example patch to obtain the SCL (scale) value. 1. Prepare the weight of the known mass. And bind its weight in grams to the constant weight node. 2. Flash the patch in the Debug mode. 3. After loading the patch you have about 20 seconds to put the prepared weight on scales. 4. Wait for the Scale value.
<a href="#">read-average</a>	Reads an average raw value from the HX711 device.
<a href="#">get-value</a>	Reads the (average raw value from the sensor - OFFSET). That is the current value without the tare weight.
<a href="#">get-weight</a>	Reads the current weight in "human readable" measure units. The units depend on the SCALE value. This is the average raw value without the tare weight and divided by the SCALE value obtained via calibration.
<a href="#">example-scale</a>	No description
<a href="#">amperka-scale</a>	Quickstart node for the scale based on Weight Sensor by Amperka.



## Test patches

The [gabbapeople/hx711](#) library provides patches for calibration (upper image) and measurement (lower image) using the HX711 device.

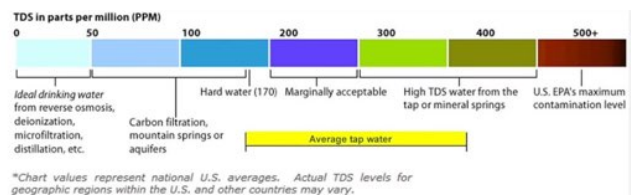
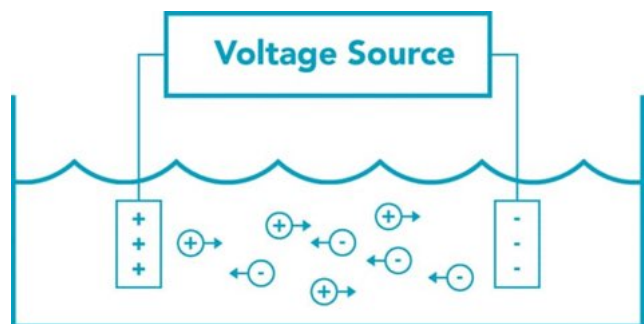


# Water quality sensor (TDS Meter V1.0)

## Description

A Total Dissolved Solids (TDS) water quality sensor is a device used to measure the concentration of dissolved solids in water, which is an important parameter for evaluating water quality in various applications, such as environmental monitoring, hydroponics, aquariums, and water treatment systems. TDS sensors typically measure the electrical conductivity (EC) of the water and use this information to estimate the TDS concentration. TDS is generally expressed in parts per million (ppm) or as milligrams per liter (mg/L). TDS is directly related to the quality of water i.e., the lower a TDS figure, the purer the water. As an example, reverse osmosis purified water will have a TDS between 0 and 10, whereas tap water will vary between 20 and 300, depending on where you live in the world.

- 1. Probes:** TDS sensors usually have two probes or electrodes that are immersed in the water sample.
- 2. Electrical conductivity measurement:** The basic principle behind TDS sensors is that the dissolved solids in water increase its electrical conductivity. The sensor applies a small voltage across the probes, which creates an electric field in the water. The dissolved ions in the water allow for the flow of electric current between the probes.
- 3. Current measurement:** The TDS sensor measures the current flowing between the probes. This current is directly proportional to the electrical conductivity of the water, which is influenced by the concentration of dissolved ions.
- 4. Temperature compensation:** The electrical conductivity of water is affected by temperature. To improve the accuracy of the TDS measurement, water temperature can be measured, and conductivity measurements corrected, according to the water temperature to provide a more accurate and consistent TDS readings.
- 5. TDS estimation:** The electrical conductivity measurement is then used to estimate the TDS concentration in the water. This is typically done using a conversion factor that relates the conductivity to the TDS concentration. The conversion factor can vary depending on the types of dissolved solids present in the water and the sensor's calibration.
- 6. Signal output and communication:** The TDS sensor outputs the TDS measurement as an analog voltage signal. The analog TDS sensor should be connected to the analog input of a microcontroller for further processing. This sensor supports 3.3 ~ 5.5V wide voltage input, and 0 ~ 2.3V analog voltage output, which makes it compatible with 5V or 3.3V Arduino boards.

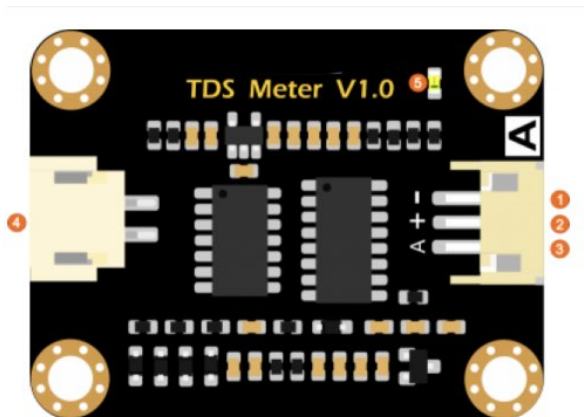
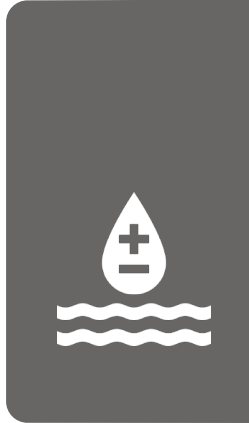


## Calibration of the sensor

An analog value that will reflect the TDS content of the liquid can be relatively easily measured. However, due to the individual differences in different TDS probes and control boards, and no onboard temperature compensation, the measured value can have some errors. Therefore, to obtain a more accurate TDS value, calibration is required before measurement. In addition, it is possible to connect a temperature sensor for temperature compensation to improve accuracy. Normally, the TDS value is half of the electrical conductivity value, that is:  $TDS = EC / 2$ .

During the calibration, liquid solutions of known electrical conductivity or TDS value is needed (e.g. 0 ppm, 500 ppm, and 1000 ppm). TDS values can also be measured using a commercial TDS pen if you do not have a standard buffer solution.

1. **Measure the sensor output:** Immerse the TDS sensor's probes in the calibration solution with the lowest TDS concentration (0 ppm, which is distilled or deionized water). Upload a simple code to the microcontroller to read the analog voltage from the sensor and display it on the serial monitor or a display device. Note down the voltage reading for this calibration point.
2. **Repeat the measurement for other calibration solutions:** Immerse the sensor's probes in the other calibration solutions with known TDS concentrations (e.g., 500 ppm and 1000 ppm). Record the analog voltage readings for each calibration point.
3. **Calculate the calibration curve:** Plot the voltage readings against the known TDS concentrations on a graph or spreadsheet. Calculate a linear regression line ( $y = mx + b$ ) that best fits the data points, where  $y$  is the TDS concentration,  $x$  is the voltage reading,  $m$  is the slope, and  $b$  is the y-intercept. This calibration curve will allow you to convert the sensor's voltage output to TDS values.
4. **Implement the calibration curve in your code:** Modify your microcontroller code to include the calibration curve parameters (slope and y-intercept). Use these parameters to convert the sensor's voltage output to TDS concentration in your application.
5. **Test the calibrated sensor:** Immerse the sensor's probes in water samples with unknown TDS concentrations and use the calibrated sensor to measure their TDS values. You can verify the accuracy of the measurements by comparing them with TDS values obtained using a commercial TDS meter or lab-grade equipment.



Num	Label	Description
1	-	Power GND(0V)
2	+	Power VCC(3.3 ~ 5.5V)
3	A	Analog Signal Output(0 ~ 2.3V)
4	TDS	TDS Probe Connector
5	LED	Power Indicator

Remember that calibration may need to be repeated periodically or when the sensor's performance changes due to aging, contamination, or other factors.

#### Specifications

1. Input Voltage: 3.3 ~ 5.5V
2. Output Voltage: 0 ~ 2.3V
3. Working Current: 3 ~ 6mA
4. TDS Measurement Range: 0 ~ 1000ppm
5. TDS Measurement Accuracy:  $\pm 10\%$  FS (25 °C)
6. TDS probe with Number of Needle: 2

#### How to Connect

The connection of TDS Sensor with Arduino is fairly simple. Connect the VCC to Arduino 5V & GND to GND. Connect its Analog pin to any analog pin of Arduino, e.g. analog pin A1 of Arduino.



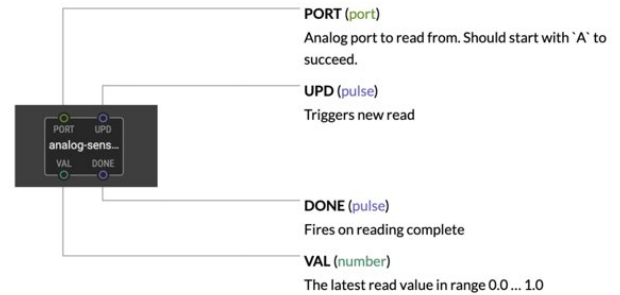
# Using the TDS Meter (V1.0)

## XOD library

Use **analog-sensor** node from **xod/common-hardware** library to read values directly from the sensor. Additional nodes must be used to estimate TDS values after calibration and to correct for different water temperatures.

As a starting point, use the following nodes:

- Capture the analog value using **analog-input** node (from the **xod/common-hardware** library): This node reads the sensor output connected to an analog pin.
- Calculate the voltage using a **multiply** node (from the **xod/math** library): This node is used to convert the ADC reading to voltage, assuming a 5V supply. Set the values of the multiply nodes: The first one should have a value of (5.0 / 1023.0) for the conversion from ADC reading to voltage.
- **subtract** node (from the **xod/math** library): This node calculates the difference between the voltage and the intercept of the calibration curve.
- **divide** node (from the **xod/math** library): This node calculates the conductivity by dividing the difference by the slope of the calibration curve. (The intercept and slope values are obtained by calibrating the TDS sensor)
- **multiply** nodes (x2) (from the **xod/math** library): These nodes calculate the TDS by multiplying the conductivity by the conversion factor and cell constant.



## wayland/total-dissolved-solids@0.0.1

License: AGPL

Library for Total Dissolved Solids (TDS) sensor with temperature compensation. A TDS sensor can be used to measure water quality. Compatible devices include <https://www.dfrobot.com/product-1662.html> and [http://www.crobot.wiki/index.php/TDS\\_\(Total\\_Dissolved\\_Solids\)\\_Meter\\_Sensor\\_SKU:\\_CQRSENTDS01](http://www.crobot.wiki/index.php/TDS_(Total_Dissolved_Solids)_Meter_Sensor_SKU:_CQRSENTDS01)

Node	Description
example-02	Patch to demonstrate use of running median to smooth analog signal from sensor. Run in debug mode.
example-03	Patch demonstrating the use of a thermometer (ds18b20) to report water temperature. Run in debug mode.
voltage-to-tds	Convert voltage measured by sensor to total dissolved solids (ppm).
example-01	Simple example patch. Run in debug mode.

## Calculating values for conversion

1. **Measure conductivity:** Immerse the sensor's probes in different calibration solutions and measure the output voltage or conductivity values.
2. **Calculate slope and intercept:** Plot the output values against the known TDS or conductivity concentrations on a graph or spreadsheet. Perform a linear regression to find the best-fit line ( $y = mx + b$ ) that relates the output values to the concentrations, where  $y$  is the TDS or conductivity concentration,  $x$  is the output value,  $m$  is the slope, and  $b$  is the  $y$ -intercept.
3. **Determine the conversion factor:** If you have calibrated the sensor using TDS values, you can assume that the conversion factor is already included in the slope. If you calibrated the sensor using conductivity values, you'll need to use a conversion factor to convert conductivity to TDS. This factor can vary depending on the composition of the dissolved solids and the type of sensor. A common conversion factor for TDS is 0.5 ( $TDS = 0.5 * conductivity$ ).
4. **Determine the cell constant:** The cell constant ( $K$ ) of a conductivity sensor is a property that describes the geometry of the sensor's electrodes and the efficiency of the sensor in measuring conductivity. You can estimate it by calibrating the sensor with a standard solution with a known conductivity and using the following formula:  $K = Conductivity / (Voltage\ output * Conversion\ factor)$ .

## XOD library: wayland/total-dissolved-solids

Matt Wayland has incorporated features into a library: **wayland/total-dissolved-solids**. The library includes a number of features such as temperature compensation if you have suitable measurement or estimation, measurement and averaging of values to reduce noise.

## Additional information

DFrobot wiki: [https://wiki.dfrobot.com/Gravity\\_\\_Analog\\_\\_TDS\\_Sensor\\_\\_Meter\\_For\\_Arduino\\_SKU\\_\\_SEN0244](https://wiki.dfrobot.com/Gravity__Analog__TDS_Sensor__Meter_For_Arduino_SKU__SEN0244)

Best Engineering Projects: <https://bestengineeringprojects.com/tds-sensor-and-arduino-interfacing/>



## An example test patch:

Simple calculation of the TDS value, based on the DFrobot Arduino IDE library:

This Arduino code snippet calculates the TDS (Total Dissolved Solids) value based on the average ADC (Analog-to-Digital Converter) value obtained from a TDS sensor.

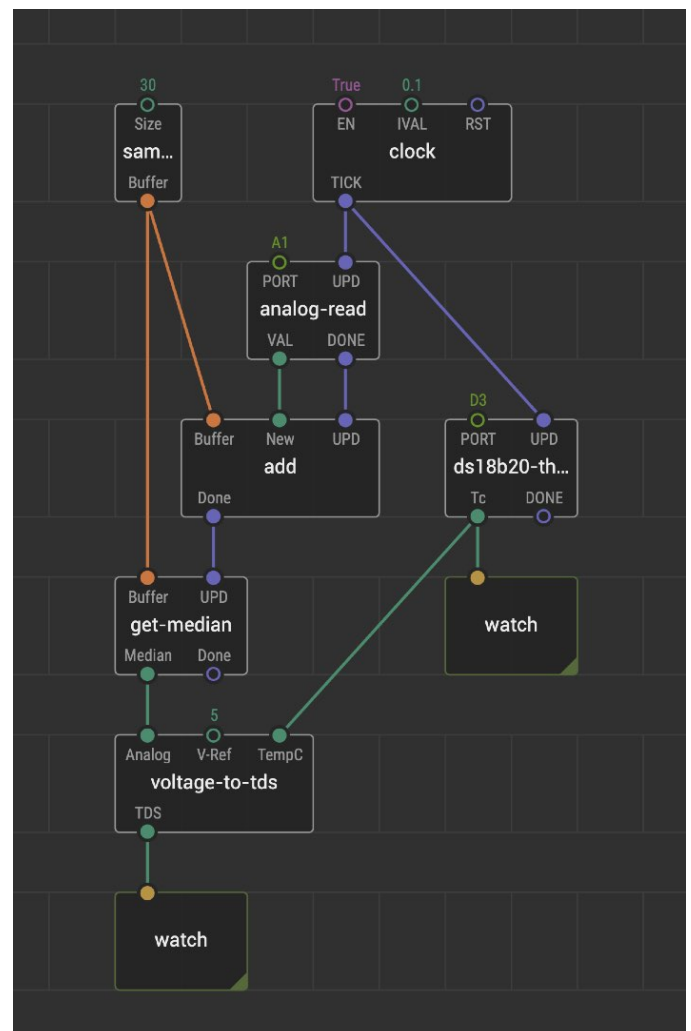
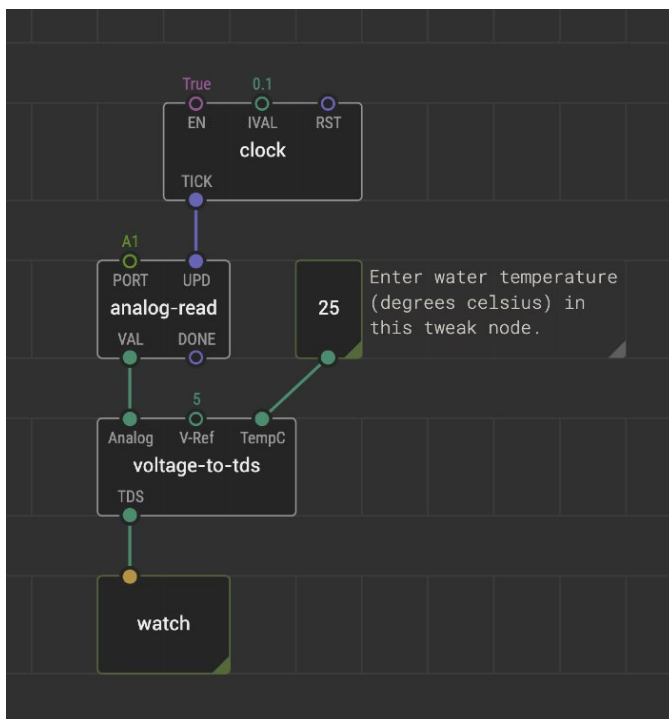
### Calculate the voltage\_value:

- Average a set of ADC values to reduce noise. (ADC = analog-digital converter)
- Multiply the average ADC value (avgval\_ADC) by the reference voltage (5.0V).
- Divide the result by the maximum ADC value for a 10-bit ADC (1024) and then divide by 6. The division by 6 is specific to the sensor's sensitivity or conversion factor.

### Calculate the TDS value:

- Use the voltage\_value in a polynomial equation to convert it to a TDS value (in ppm, or parts per million).
- The equation is:  $TDS = (133.42 * voltage\_value^3 - 255.86 * voltage\_value^2 + 857.39 * voltage\_value) * 0.5$
- The coefficients (133.42, -255.86, and 857.39) are derived from the TDS sensor's calibration data.

The code calculates the TDS value in water based on the sensor's voltage readings, converting the voltage to TDS using a polynomial equation. (The same parameters are used in the Seed Studio library for a similar sensor).



# Water level sensor

## Description

A water level sensor is a device that measures the level or depth of water in a container or natural body of water.

Resistive water level sensors use the change in electrical resistance that occurs when a conductive material (usually a strip or probe) comes into contact with water. The sensor has two conductive strips or probes at different heights or positions, and when water touches these probes, it creates an electrical connection between them.

There are several parallel copper tracks on the pcb. Those are alternately connected to VCC and GND. While the sensor is dry there is no electrical connection between them but as the water level is rising they get connected. We can treat them as a variable resistor.

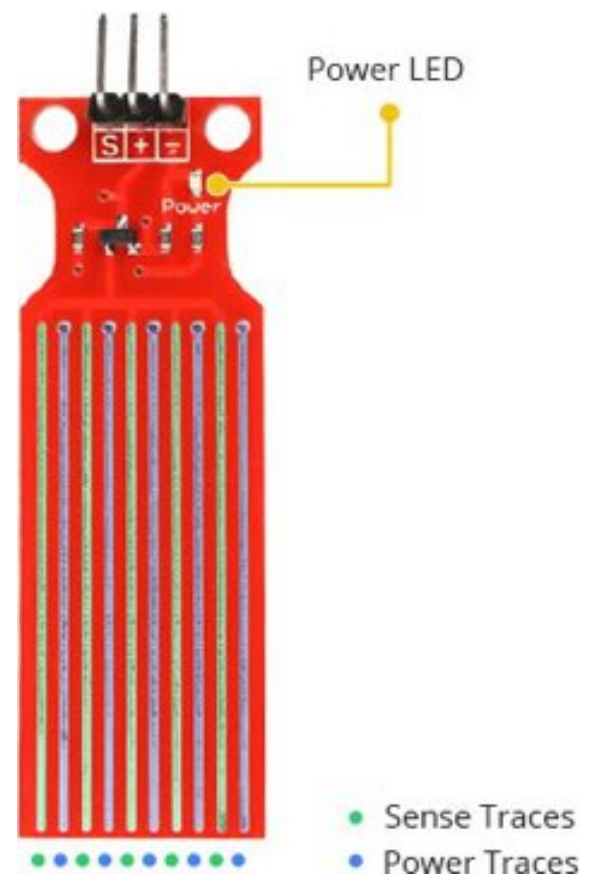
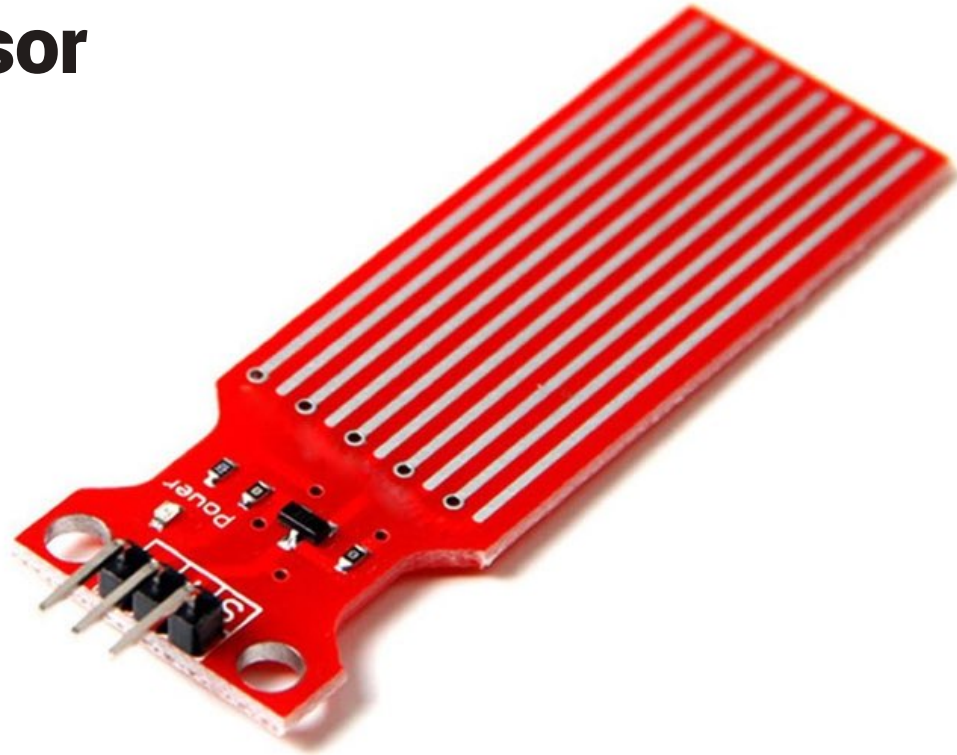
The water level sensor / leak detection sensor is a 3-pin module that outputs an analogue signal (generally 0 to 500) that indicates the approximate depth of water submersion. When used in conjunction with a pull up resistor, it can be used as a digital device to indicate the presence or water.

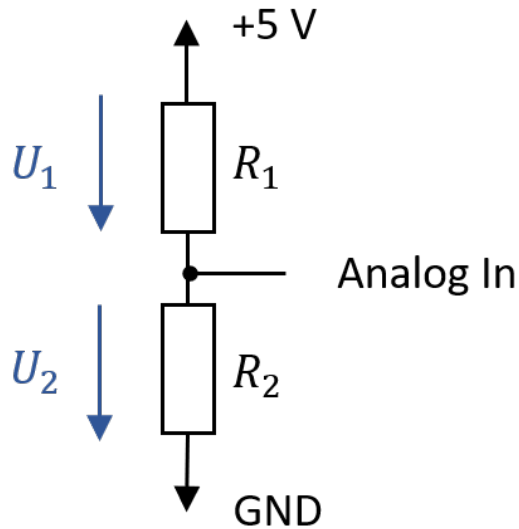
This sensor has a series of ten exposed copper traces, five of which are power traces and five are sense traces. These traces are interlaced so that there is one sense trace between every two power traces. Usually these traces are not connected but are bridged by water when submerged. The change in resistance corresponds to the distance from the top of the sensor to the surface of the water. The resistance is inversely proportional to the height of the water:

- The more water the sensor is immersed in, results in better conductivity and will result in a lower resistance.
- The less water the sensor is immersed in, results in poor conductivity and will result in a higher resistance.

The higher the water level rises, the lower drops the resistance. The resistance can be measured using a voltage divider as shown in the picture above. It's a really simple circuit with R1 being a fixed resistor and R2 being our variable resistor. Since the current through all resistors in a row is the same, the voltage drop at every resistor is dependent on it's resistance - or it's proportion of the total resistance of the circuit. So as the water level increases, the voltage measured at 'Analog In' increases until the whole sensor is covered in water. With the total height of the sensor as well as the max and min values at 'Analog In', we can calculate the water level. Resistive sensors are simple, low-cost, and easy to use but may be affected by the conductivity of the water.

(Thank you to <https://lastminuteengineers.com> for some of the information on this page)





## Specifications

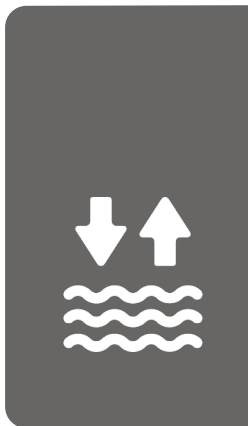
Operating Voltage: +5V  
 Working Current : <20mA  
 Sensor Type : Analog or Digital  
 Water Detection Area : 1.58in X .63in (40mm X 16mm)  
 Mounting Hole Size : 0.12in (3mm)  
 Operating Humidity: 10% to 90% (non-condensing)  
 Working Temperature: (-30 to 50 degrees C)

## How to Connect

The 3 pins to connect to the Grove board on the module are:

- S (Signal) pin** is an analog output that will be connected to one of the analog inputs on your Arduino.
- + (VCC) pin** supplies power for the sensor. It is recommended that the sensor be powered with 3.3V – 5V. Please note that the analog output will vary depending on what voltage is provided for the sensor.
- (GND)** is a ground connection.

First you need to supply power to the sensor. For that you can connect the + (VCC) pin on the module to 5V on the Arduino and – (GND) pin to ground. However, one issue with these sensors is their short lifespan when exposed to a moist environment. Having power applied to the probe constantly speeds the rate of corrosion significantly. To overcome this, we recommend that you do not power the sensor constantly, but power it only when you take the readings. An easy way to accomplish this is to connect the VCC pin to a digital pin of an Arduino and set it to HIGH or LOW as per your requirement. Connect the S (Signal) pin to a suitable unused analogue pin on the Arduino board.



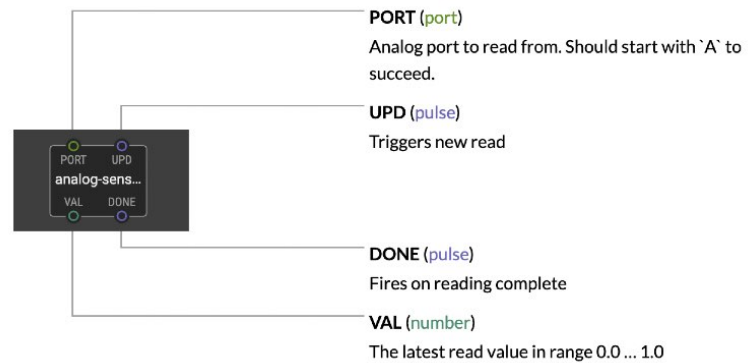
# Using a water level sensor

## XOD library

Use **analog-sensor** node from **xod/common-hardware** library. Values can be read repeatedly from the analogue port at a suitable interval and used for calibration, display or further calculations. If a digital port is being used to power the sensor (to allow intermittent powering of the device, and minimise electrolytic corrosion) - the relevant port can be turned on, and a delay of >10 mSec added before reading the sensor and turning off the digital port again.

## Calibration

To get accurate readings out of your water level sensor, it is recommended that you first calibrate it for the particular type of water that you plan to monitor. Pure water is not conductive, minerals and impurities in water make it conductive. So, your sensor may be more or less sensitive depending on the type of water you use. Before you start storing data or triggering events, you should see what readings you are actually getting from your sensor. Note what values your sensor outputs when the sensor is completely dry -vs- when it is partially submerged in the water -vs- when it is completely submerged. In the example below, you see the following values in the serial monitor when the sensor is dry (0) and when it is partially submerged in the water (~420) and when it is completely submerged (~520). With a series of measurements, it is possible to draw a calibration curve that allows prediction of the precise physical depth of the water.

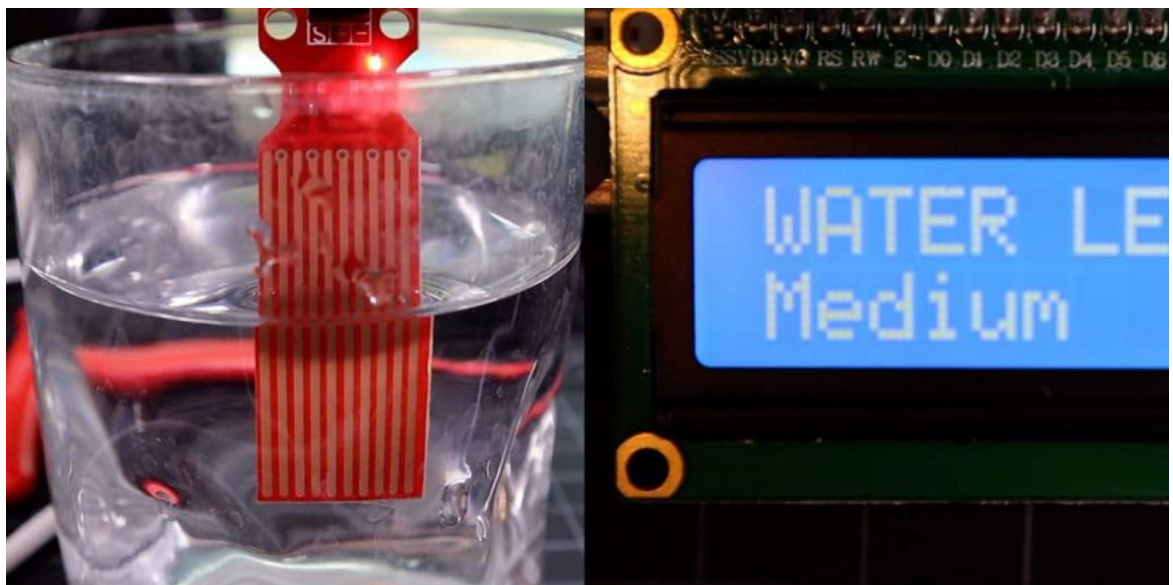


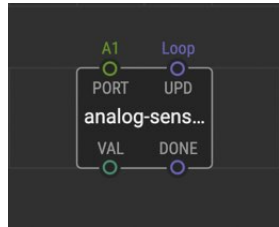
## Further information:

**Water sensor hookup:** <https://www.hackster.io/NewMC/water-level-monitor-b42be9>

**How it works and interfacing with Arduino:** <https://lastminuteengineers.com/water-level-sensor-arduino-tutorial/>

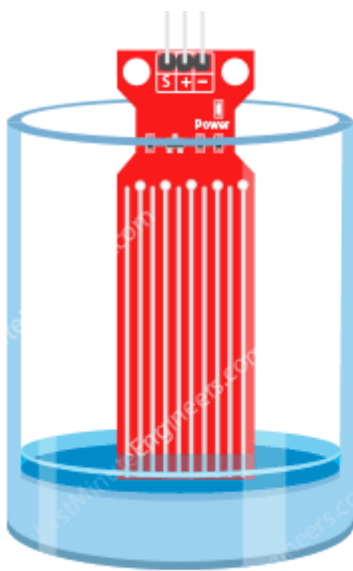
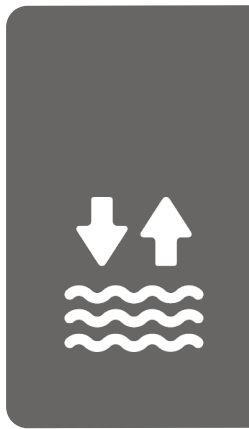
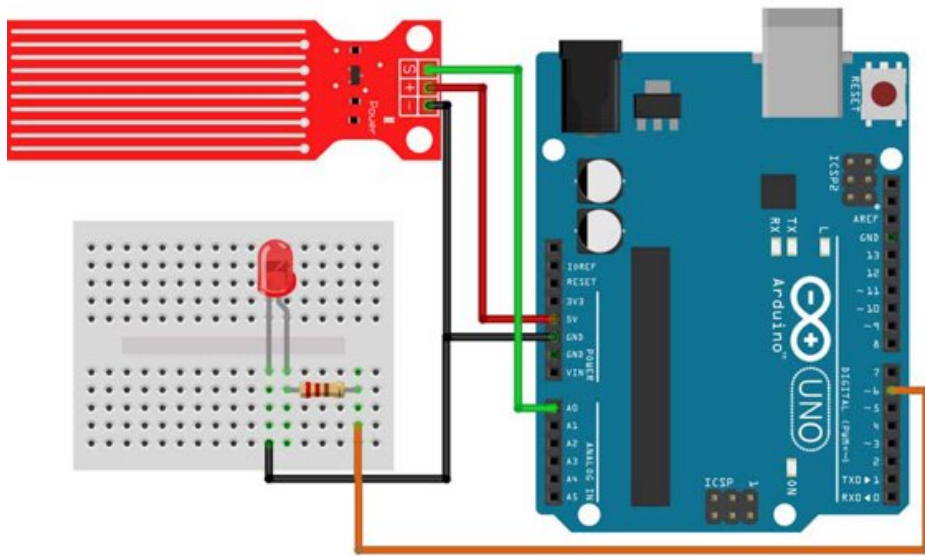
**Water level sensor tutorial:** <https://www.thegeekpub.com/236571/arduino-water-level-sensor-tutorial/>



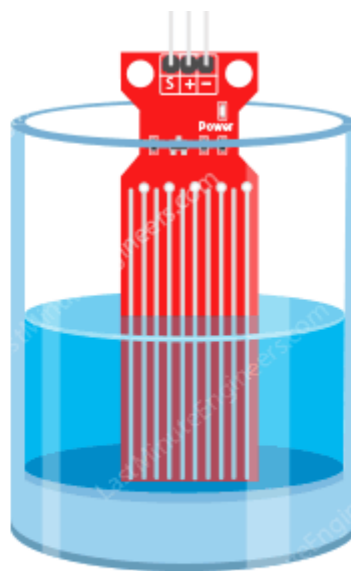


### Test patch

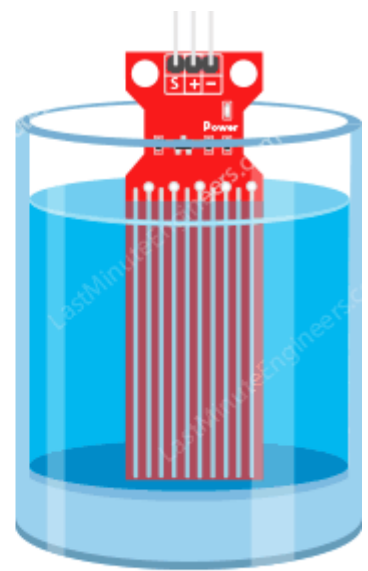
Set port to D2 (or whichever port you are using) and UPD to 'Continuously'. Connect SIG to a watch node. Upload and debug.



Status: Dry  
Test Reading: ~0



Status: Partially submerged  
Test Reading: ~420



Status: Fully submerged  
Test Reading: ~520



# Capacitive soil moisture sensor

## Description

A capacitive soil moisture sensor measures the moisture content in the soil based on changes in capacitance. The sensor typically consists of a probe with conductive plates, which form a capacitor. The probe is inserted into the soil, and the soil acts as the dielectric material between the conductive plates.

The capacitance of a capacitor depends on the surface area of the conductive plates, the distance between the plates, and the dielectric constant of the material between the plates. A capacitive moisture sensor works by measuring the changes in capacitance caused by the changes in the dielectric. It does not measure moisture directly (pure water does not conduct electricity well), instead it measures the ions that are dissolved in the moisture. These ions and their concentration can be affected by a number of factors, for example adding fertilizer for instance will decrease the resistance of the soil. Capacitive measuring basically measures the dielectric that is formed by the soil, and the water is the most important factor that affects the dielectric. The dielectric constant of the soil changes with its moisture content. When the soil is dry, its dielectric constant is low, and when the soil is wet, its dielectric constant increases.

The sensor measures the capacitance between the conductive plates, which in turn reflects the dielectric constant of the soil. By calibrating the sensor to the specific soil type and knowing the relationship between the dielectric constant and soil moisture content, the moisture level in the soil can be determined.

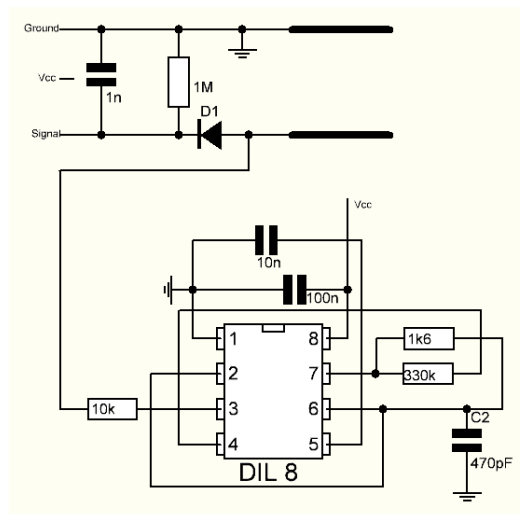
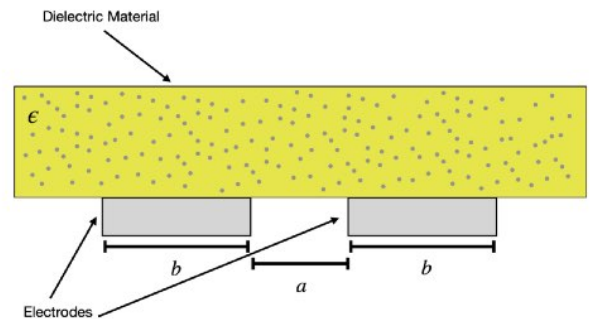
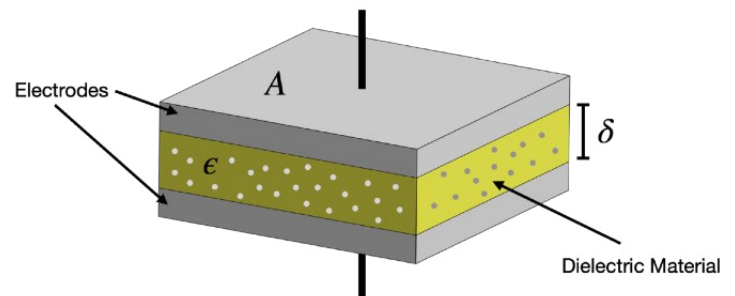
Capacitive soil moisture sensors offer several advantages over resistive sensors, such as:

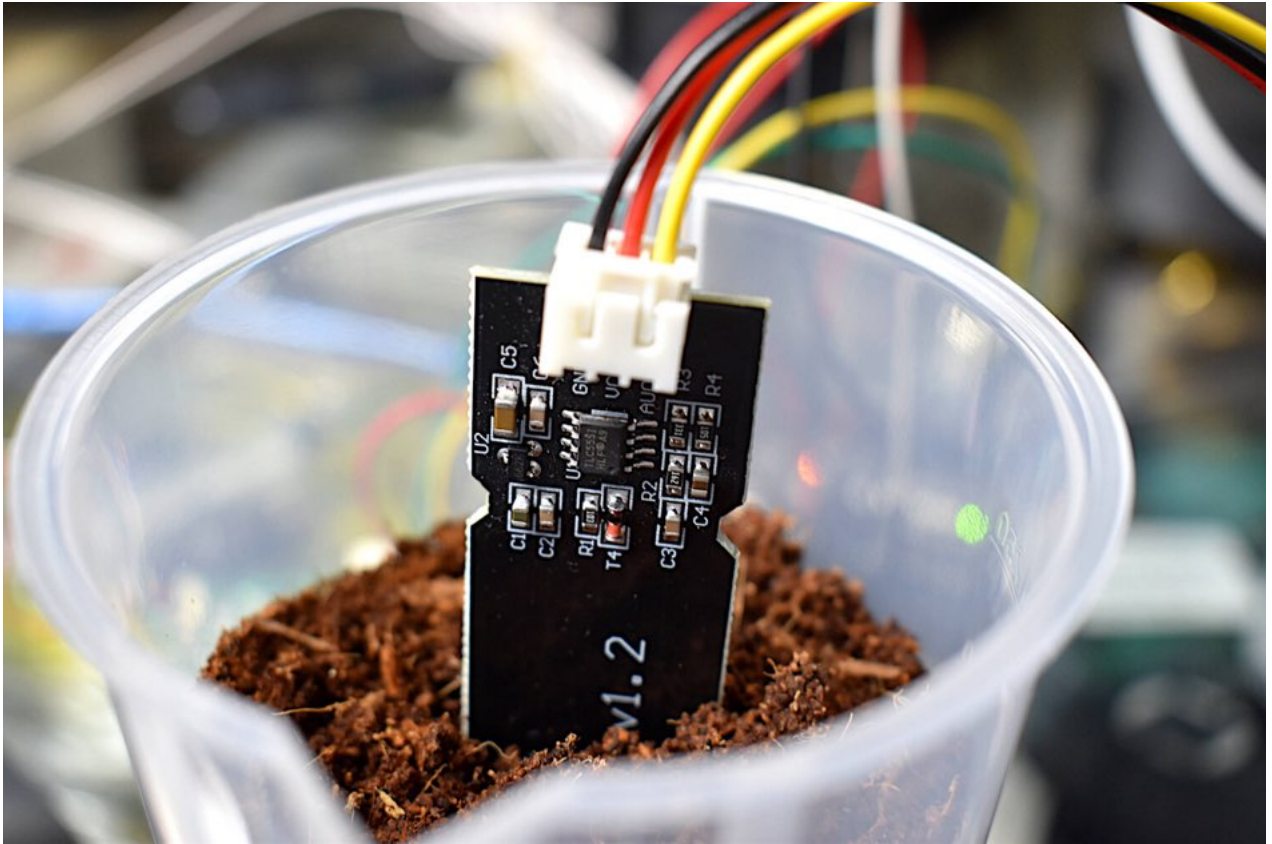
- 1. Non-contact measurement:** The capacitive sensor doesn't rely on direct electrical contact with the soil, reducing the risk of corrosion and improving the sensor's lifespan.
- 2. Less sensitive to soil salinity:** Capacitive sensors are less affected by the soil's conductivity, making them more reliable in various soil types.
- 3. Lower power consumption:** Capacitive sensors typically consume less power, making them suitable for battery-powered or energy-harvesting applications.

Traditional soil moisture sensors are prone to corrosion with a limited lifespan regardless of measures taken. This capacitive soil moisture sensor features no exposed metal. The result is a more robust sensor without corrosion worries. However, capacitive soil moisture sensors may require calibration for different soil types, and their accuracy can be affected by factors such as temperature and probe positioning.

## Electronics

Hardware schematic for capacitive soil moisture sensor  
Soil and sensor form a capacitor where the capacitance varies according to the water content present in the soil. The capacitance is converted into voltage level basically from 1.2V to 3.0V maximum. There is a fixed frequency oscillator that is built with a 555 Timer IC. The square wave generated is then fed to the sensor like a capacitor. To a square wave signal that capacitor, however, has a certain reactance, or for argument's sake a resistance that forms a voltage divider with a pure ohm type resistor (the 10k one on pin 3). The greater is the soil moisture,





the higher the capacitance of the sensor. Consequently, there is a smaller reactance to the square wave, thus lowering the voltage on the signal line. The voltage on the Analog signal pin can be measured by an analog pin on the Arduino which represents the humidity in the soil.

## Specifications

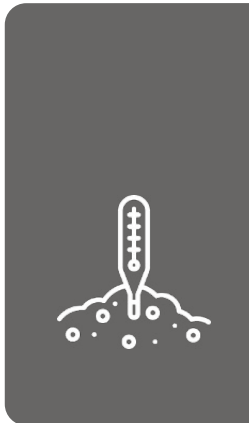
Using the capacitive sensor with an Arduino is simple as the Arduino has a built-in 10 bit ADC. The sensor has a built-in voltage regulator that supports 3.3V, meaning it can work with a 3.3V development board as well as 5V. It is recommended that 3.3V power be used if available, to maximise resolution of the ADC.

- Sensor type: Analog
- Operating voltage: 3.3 VDC
- Output voltage: 0-3.0 VDC
- Interface: PH2.54 3-pin

## How to Connect

To connect to the Grove board:

- Connect top plug wire (GND) to (GND)
- Connect second plug wire (VCC) to 3.3V
- Connect bottom plug wire (AOUT) to an analogue port on the Arduino board (e.g. A1)



# Using a capacitive soil moisture sensor

## XOD library

Use `analog-sensor` node from `xod/common-hardware` library.

## Calibration

There are some excellent tutorials with guides to the theory, use and calibration of this capacitive soil moisture sensor, for example:

- <https://makersportal.com/blog/2020/5/26/capacitive-soil-moisture-calibration-with-arduino>
- <https://www.instructables.com/Soil-Moisture-Sensor-Calibration/>
- <https://create.arduino.cc/projecthub/wteele/auto-calibration-program-for-capacitive-soil-moisture-sensor-066070>

One technique is to use a gravimetric technique to calibrate capacitive-type electromagnetic soil moisture sensors. Capacitive soil moisture sensors exploit the dielectric contrast between water and soil, where dry soils have a relative permittivity between 2-6 and water has a value of roughly 80. Accurate measurement of soil water content is essential for applications in agronomy and botany - where the under- and over-watering of soil can result in ineffective or wasted resources. With water occupying up to 60% of certain soils by volume, depending on the specific porosity of the soil, calibration must be carried out in every environment to ensure accurate prediction of water content. An Arduino can be used to read the analog signal from the capacitive sensor, which can be calibrated to volumetric soil moisture content via gravimetric methods (using volume and weight of dry and wet soil). Alternatively there are autocalibration approaches.

The capacitance of the sensor is measured by means of a 555 based circuit that produces a voltage proportional to the capacitor inserted in the soil. One can then measure this voltage by use of an Analog to Digital Converter (ADC), which produces a number that we can then interpret as soil moisture. The final output value is affected by probe insertion depth and how tight the soil packed around it is. Value\_1 is the value for dry soil and Value\_2 is the value for saturated soil. For example: Value\_1 = 520; Value\_2 = 260.

The range will be divided into three sections: dry, wet, water. Their related values are:

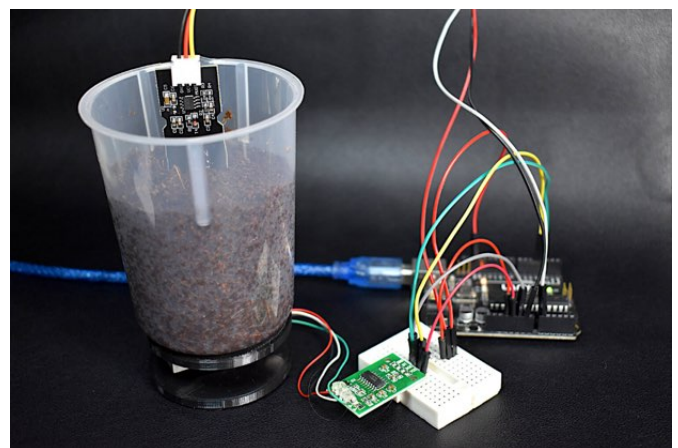
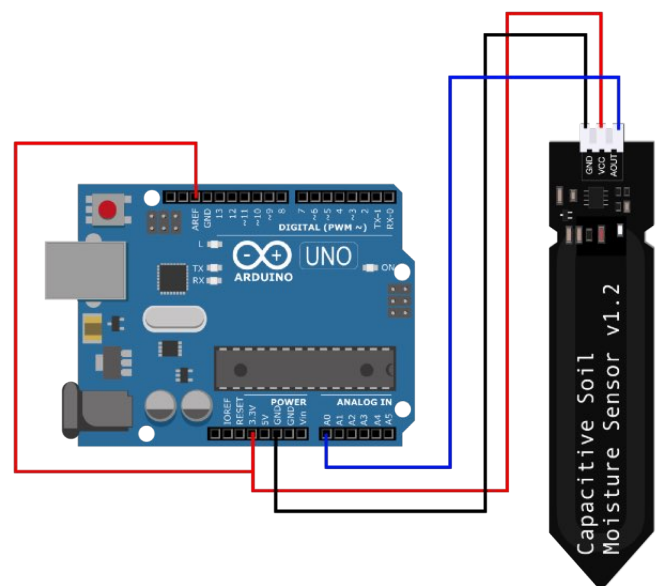
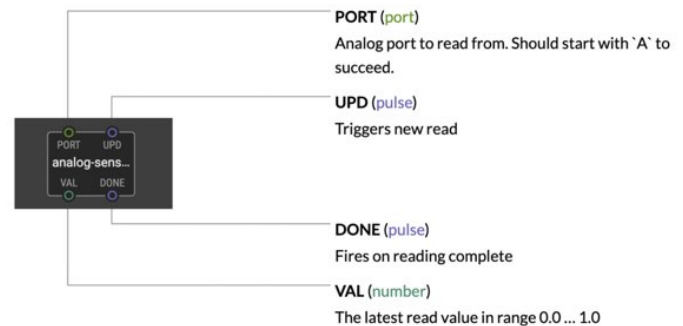
- Dry: (520 430]
- Wet: (430 350]
- Water: (350 260]
- 

Assuming linearity, you can convert these values to a "percent" of water. Just remember, in reality, Dry is not 0% moisture and "Water" may not be 100% moisture, at least at the lower values. Still, it is a useful measurement. (modified from: <https://www.switchdoc.com/2020/06/tutorial-capacitive-moisture-sensor-grove/>)

## Further information:

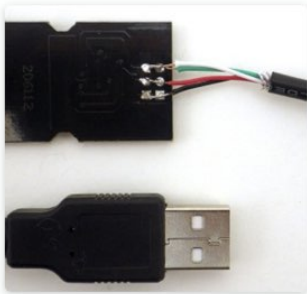
Sensor use and calibration: <https://how2electronics.com/interface-capacitive-soil-moisture-sensor-arduino/>

Theory and calibration: <https://makersportal.com/blog/2020/5/26/capacitive-soil-moisture-calibration-with-arduino>



Weather/environmental sealing of the capacitive soil moisture sensor: <https://thecavepearlproject.org/2020/10/27/hacking-a-capacitive-soil-moisture-sensor-for-frequency-output/>. The article also describes hacking the device for frequency output. **Comparison of soil sensors and some tips:** <https://arduino diy.wordpress.com/2020/08/>

## Method for permanent waterproof encapsulation of electronics



Clean the sensor with 90% isopropyl alcohol. An old USB or phone cable works for light – duty sensor applications. Here I've combined two of the four wires to carry the output.



3/4"(18mm) to 1"(24mm), 2:1 heat-shrink forms a container around the circuits, with smaller **adhesive lined** 3:1 at the cable to provide a seal to hold the liquid epoxy.



Only fill to about 15% of the volume with epoxy. Here I'm using Loctite E30-CL. This epoxy takes 24 hours to fully cure.



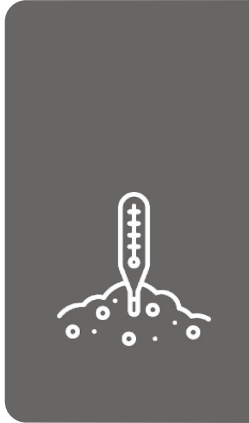
**GENTLE heating** compresses the tubing **from the bottom up**. This forces the epoxy over the components



Before finishing, use a cotton swab to seal the edges of the probe with the epoxy. Cut PCBs can absorb several % water if edges are left exposed.



Complete the heating over a rubbish bin to catch the overflow & wipe the front surface of the probe so that you don't have any epoxy on the sensing surfaces.









# Software Expansion

## Finding XOD Nodes

Once you have found a component that you would like to use you will need to find a XOD node to represent that hardware. The **xod/common-hardware** library provides nodes for a number of commonly used components and many more nodes have been created by the XOD community.

As a standard electronic components are assigned a 'reference designator'. This short combination of letters and numbers identifies specific components, for example, the Grove board uses the 'BMP280' barometer and the 'SSD1306' OLED screen. This system is useful as it is easy to compare components, understand what hardware others are using, and search for complimentary nodes and/or code for programming the hardware.

In XOD, many contributors have created specific libraries to deal with certain pieces of hardware, and you can search these libraries on the XOD website at [www.xod.io/libs](http://www.xod.io/libs). It is usually easiest to search using the hardware's reference designator. Another useful way to find libraries is to search the XOD forum at [www.forum.xod.io](http://www.forum.xod.io). This can help you to find relevant node and libraries, as well as identify any common issues others have had when using specific pieces of hardware.

## Using Arduino IDE

XOD, and other 'no-code' programming options, are a really useful way to get started working with microcontrollers. They require less up-front learning and offer an alternative and intuitive way of thinking about programming. However, you may also be interested in learning to code - either as an extension to your XOD programming skills, or as an alternative. Arduino provides its own free software for programming: the Arduino IDE, which is available for download at [www.arduino.cc/en/software](http://www.arduino.cc/en/software). This software uses the C++ language for programming. Grove provides an excellent guide for programming your board using the Arduino IDE at [www.files.seeedstudio.com/wiki/Grove-Beginner-Kit-For-Arduino/res/Grove-Beginner-Kit-For-ArduinoPDF.pdf](http://www.files.seeedstudio.com/wiki/Grove-Beginner-Kit-For-Arduino/res/Grove-Beginner-Kit-For-ArduinoPDF.pdf).

One advantage of using the Arduino IDE is the vast amount of resources available for working with almost any piece of hardware. Whilst the XOD community is growing fast and new libraries are being added all the time, you may find that certain hardware and components do not yet have compatible XOD nodes. In this case, there is almost always an Arduino IDE library that can be used. Another option is to convert existing Arduino IDE libraries into XOD libraries. Matt Wayland has written an excellent guide detailing how to convert Arduino libraries for use in XOD, available at [www.biomaker.org/s/converting-arduino-to-xod\\_wayland.pdf](http://www.biomaker.org/s/converting-arduino-to-xod_wayland.pdf). Further guidance on creating libraries of XOD in C++ is available on the XOD website at [www.xod.io/docs/guide/nodes-for-xod-in-cpp](http://www.xod.io/docs/guide/nodes-for-xod-in-cpp) and [www.xod.io/docs/guide/analog-sensor-node](http://www.xod.io/docs/guide/analog-sensor-node).

You can also use XOD in combination with Arduino IDE. For example, you could write a programme in XOD, then navigate to 'Deploy > Show Code for Arduino' in the menu bar to export this programme in code. You could then add additional code in Arduino. For example, code to control a device not supported in XOD.

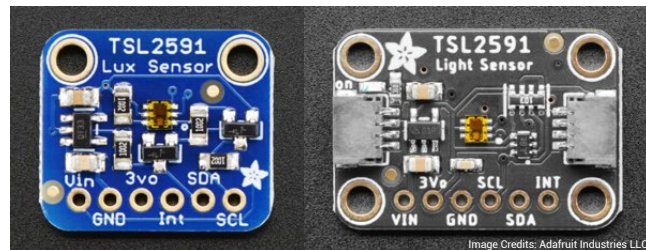


# Converting Arduino Libraries to XOD

## Why Convert Arduino Libraries?

Whilst the XOD community is growing fast and new libraries are being added all the time, you may find that certain hardware and components do not yet have compatible XOD nodes. In this case, there is almost always an Arduino IDE library that can be used. Arduino libraries exist for a huge range of breakout boards and other devices (see [www.arduino-libraries.info](http://www.arduino-libraries.info)). If you have a little C++ experience, it is easy to incorporate these libraries into XOD.

If you cannot find a XOD library for a device, you will need to look for a class-based Arduino library. Manufacturers of breakout boards typically provide C++ libraries for their devices. On the product pages of companies such as Adafruit you will typically find links to code repositories. For more unusual devices a web search will often find libraries developed by hobbyists.



*The Adafruit Industries TSL2591 Lux Sensor Breakout Boards. Left to right: solderable and STEMMA-QT versions.*

In this tutorial we will create a XOD library for the TSL2591 high dynamic range digital light sensor. Adafruit produce breakout boards for this sensor, available as either a solderable version or with a STEMMA-QT socket (Qwiic-compatible, not Grove-compatible). You can learn more about these breakout boards at [learn.adafruit.com/adafruit-tsl2591](http://learn.adafruit.com/adafruit-tsl2591).

Adafruit's code repository for their TSL2591 library is available on github at [www.github.com/adafruit/Adafruit\\_TSL2591\\_Library](https://www.github.com/adafruit/Adafruit_TSL2591_Library). You should download this library before starting this tutorial.

Please note that this tutorial assumes a basic knowledge of the XOD IDE and C++. For our Beginner's Guide to XOD see [www.biomaker.org/nocode-programming-for-biology-handbook](http://www.biomaker.org/nocode-programming-for-biology-handbook). For an excellent beginner's short course in C++ see [www.codecademy.com/learn/learn-c-plus-plus](http://www.codecademy.com/learn/learn-c-plus-plus).

# Creating a XOD library for the TSL2591 Lux sensor

## Requirements

- Computer running MacOS, Windows or Linux with XOD software and USB driver installed (required)
- Adafruit TSL2591 library (required - download from [www.github.com/adafruit/Adafruit\\_TSL2591\\_Library](https://www.github.com/adafruit/Adafruit_TSL2591_Library))
- Arduino IDE (for testing - download from [www.arduino.cc/en/Main/Software](https://www.arduino.cc/en/Main/Software))
- Arduino board and USB connector cable (for testing)
- Adafruit TSL2591 Lux Sensor and connectors (for testing)

When presented with a new device the first thing you should do is check if it is already supported in XOD. There is a searchable database of XOD libraries at [www.xod.io/libs](https://www.xod.io/libs).

If you search for "light sensor" or "TSL2591" you will find that a library already exists for this device ([www.xod.io/libs/wayland/tsl2591-light-sensor](https://www.xod.io/libs/wayland/tsl2591-light-sensor)). However, for the purposes of this tutorial, we will pretend that there is no library for the TSL2591, and will instead convert the Adafruit C++ library for use in XOD.

It is a good idea to test libraries you find using the Arduino IDE. Well written libraries will include example sketches. Reading through the sketches can help you to understand how the methods in the library are used.

In this tutorial we will first test the Adafruit TSL2591 library in the Arduino IDE, then 'wrap' this library for use in XOD. First we will create a device node to represent the TSL2591 sensor, then we will create action nodes to represent each of the library's member functions.

## Connecting the TSL2591 Sensor to your Arduino

### CONNECTING THE SOLDERABLE SENSOR TO A GROVE BOARD:

- Solder a six pin header set to the breakout board
- Plug a 4-pin Grove-to-female connector into an I2C socket on the Grove board
- Fit the male header pins on the breakout board into the female connector ends.
- Make sure the wire colours match the pin labels as follows: black to GND, red to Vin, white to SDA, yellow to SCL.

### CONNECTING THE SOLDERABLE SENSOR TO A DIFFERENT ARDUINO BOARD:

- Solder a six pin header set to the breakout board
- Use male-to-female wires to connect the header pins on the breakout board to the header sockets on the board
- Make sure the wires are connected as follows: GND to GND, Vin to VIN, SDA to SDA and SCL to SCL.

### CONNECTING THE STEMMA-QT SENSOR:

- Use a SparkFun Qwiic Arduino board and connect with Qwiic cables.
- Or connect a SparkFun Qwiic shield to any other Arduino board and connect with Qwiic cables.



# Testing the Arduino library

**1**



arduino.cc/en/software/

**DOWNLOAD OPTIONS**

Windows Win 7 and newer  
Windows ZIP file

Windows app Win X.1 or 10 **Get**

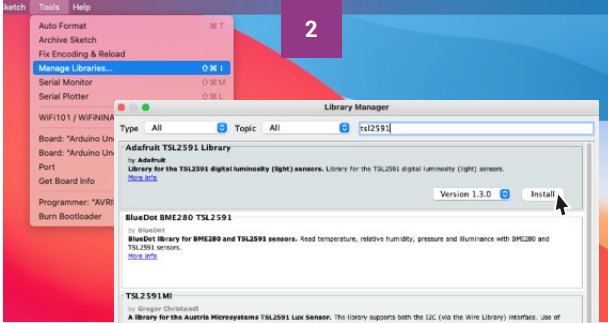
Linux 42 bits  
Linux 64 bits  
Linux ARM 32 bits  
Linux ARM 64 bits

Mac OS X 10.10 or newer

**INSTALL ARDUINO IDE**

Download the Arduino IDE from the Arduino website and install it on your computer.

**2**



Tools Help

Auto Format  
Archive Sketch  
File Encoding & Reload  
**Manage Libraries...** OK I  
Serial Monitor  
Serial Plotter

Board: "Arduino Uno"  
Board: "Arduino Uno"  
Port  
Get Board Info  
Programmer: "AVRISP" Burn Bootloader

Library Manager

Type: All Topic: All [tsl2591]

Adafruit TSL2591 Library  
by Adafruit  
Library for the TSL2591 digital luminosity (light) sensors. Library for the TSL2591 digital luminosity (light) sensors.  
[View file](#)

Version 1.3.0 **Install**

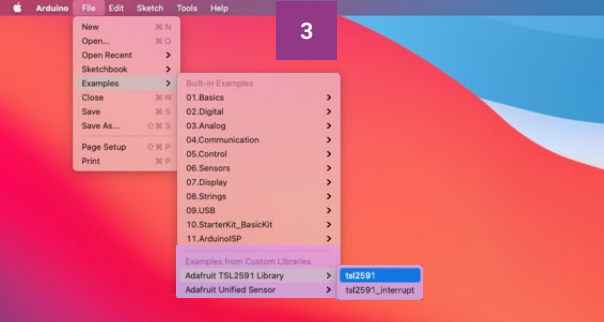
BlueDot BME280 TSL2591  
BlueDot library for BME280 and TSL2591 sensors. Read temperature, relative humidity, pressure and illuminance with BME280 and TSL2591 sensors.  
[View file](#)

TSL2591.MI  
by Kings Christwell  
A library for the Austria Microsystems TSL2591 Lux Sensor. The library supports both the I2C (via the Wire Library) interface. Use of

**ADD LIBRARY TO IDE**

From the 'Tools' menu select 'Manage Libraries'. In the Library Manager search for 'tsl2591'. Select the most recent version of the Adafruit TSL2591 Library and click 'Install'. If you receive a prompt informing you that the library is dependent on other libraries, click 'Install all'.

**3**



File Edit Sketch Tools Help

New ⌘ N  
Open... ⌘ O  
Open Recent  
Sketchbook  
Examples  
Close ⌘ W  
Save ⌘ S  
Save As... ⌘ S  
Page Setup ⌘ P  
Print

Built-in Examples

- 01. Basics
- 02. Digital
- 03. Analog
- 04. Communication
- 05. Control
- 06. Sensors
- 07. Display
- 08. Strings
- 09. USB
- 10. StarterKit\_BasicKit
- 11. ArduinoISP

Examples from Custom Libraries

- Adafruit TSL2591 Library
- Adafruit Unified Sensor
- tsl2591**
- tsl2591\_interrupt

**RUN AN EXAMPLE SKETCH**

Running an example sketch is a good way of checking that the device is wired correctly to the Arduino board, that the device is working, and that the library is working. Open an example sketch by navigating to 'File' > 'Examples' > 'Adafruit TSL2591 Library' > 'tsl2591'.

**4**



tsl2591 | Arduino 1.8.13

Upload

```
tsl2591.h
/* TSL2591 Digital Light Sensor */
/* Dynamic Range: 600M:1 */
/* Maximum Lux: 85K */

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_TSL2591.h"

// Example for demonstrating the TSL2591 libra

// connect SCL to I2C Clock
// connect SDA to I2C Data
// connect VCC to 3.3-5V DC
```

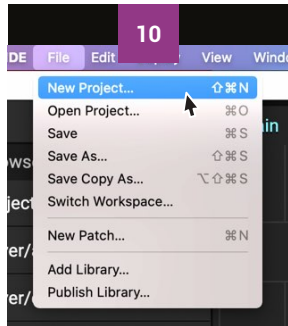
**UPLOAD**

Click on the Upload button. This is the button on the top left of the screen that looks like an arrow.



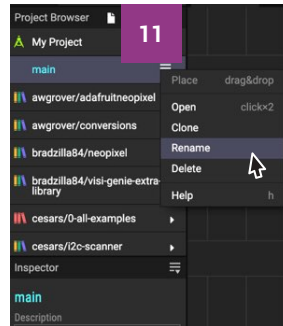


# Creating a TSL2591-device node



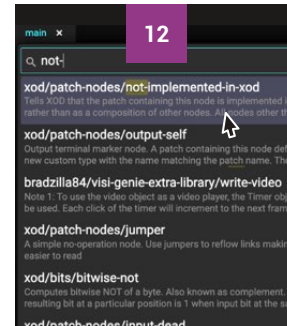
## CREATE A NEW PROJECT IN XOD

Open the XOD software and start a new project by navigating to 'File' > 'New Project...!.



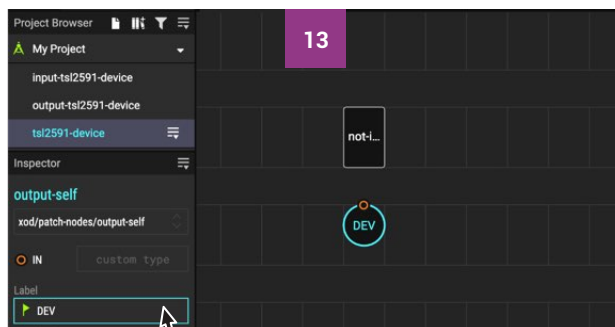
## RENAME YOUR PATCH

Right-click on the **main** patch in the Project Browser and select 'Rename'. Name the patch '**tsl2591-device**'.



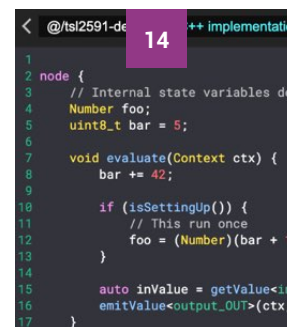
## NOT-IMPLEMENTED-IN-XOD NODE

Double click on the patch and type '**not-implemented-in-xod**'. When the node appears, click to add it to the patch.



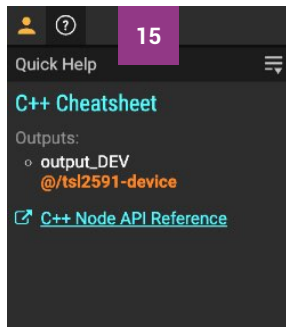
## ADD AN OUTPUT-SELF NODE

Add an **output-self** node (**xod/patch-nodes**) in the same way. Use the 'Label' box of the Inspector Pane to name it 'DEV'. When you do this, you should notice two new patches will automatically appear in the Project Browser: **input-tsl2591-device** and **output-tsl2591-device**.



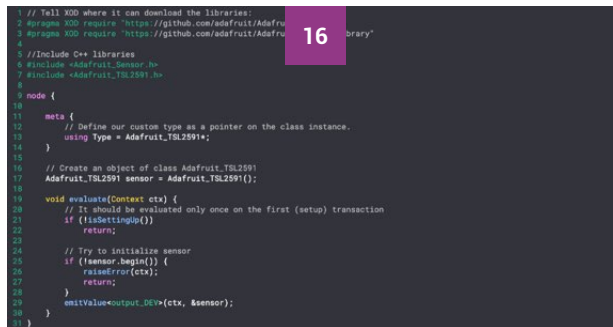
## OPEN C++ CODE EDITOR

Double-click on the **not-implemented-in-xod** node to open the C++ code editor. You will see some template code in the editor.



### QUICK HELP

If you look at the Quick Help pane there is a C++ Cheatsheet listing terminal nodes in the patch. Here there is a single output node.



### ADD CODE

Delete the template code and add the prepared C++ code (see below).

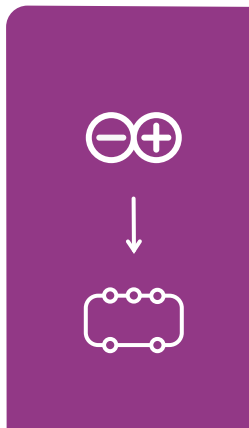
### C++ CODE FOR XOD TSL2591 DEVICE

```
// Tell XOD where it can download the libraries:
#pragma XOD require "https://github.com/adafruit/Adafruit_Sensor"
#pragma XOD require "https://github.com/adafruit/Adafruit_TSL2591_Library"

//Include C++ libraries
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2591.h>

node {

  meta {
    // Define our custom type as a pointer on the class instance.
    using Type = Adafruit_TSL2591*;
  }
  // Create an object of class Adafruit_TSL2591
  Adafruit_TSL2591 sensor = Adafruit_TSL2591();
  void evaluate(Context ctx) {
    // It should be evaluated only once on the first (setup) transaction
    if (!isSettingUp())
      return;
    // Try to initialize sensor
    if (!sensor.begin()) {
      raiseError(ctx);
      return;
    }
    emitValue<output_DEV>(ctx, &sensor);
  }
}
```



# The TSL2591-device C++ code

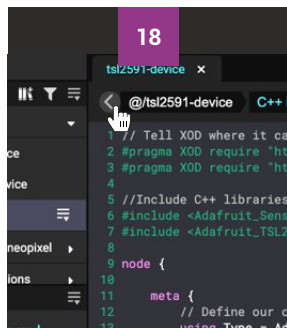
```

< @/tsl2591-device C++ implementatio 17
A 1 // Tell XOD where it can download the libraries:
2 #pragma XOD require "https://github.com/adafruit/Adafruit_Sensor"
3 #pragma XOD require "https://github.com/adafruit/Adafruit_TSL2591_Library"
4
5 //Include C++ libraries
6 #include <Adafruit_Sensor.h> B
7 #include <Adafruit_TSL2591.h>
8
9 node {
10
11     meta {
12         // Define our custom type as a pointer on the class instance.
13         using Type = Adafruit_TSL2591*;
14     }
15
16     // Create an object of class Adafruit_TSL2591
17     Adafruit_TSL2591 sensor = Adafruit_TSL2591(); D
18
19     void evaluate(Context ctx) {
20         // It should be evaluated only once on the first (setup) transaction
21         if (!isSettingUp())
22             return;
23
24         // Try to initialize sensor
25         if (!sensor.begin()) {
26             raiseError(ctx);
27             return;
28         }
29         emitValue<output_DEV>(ctx, &sensor); F
30     }
31 }

```

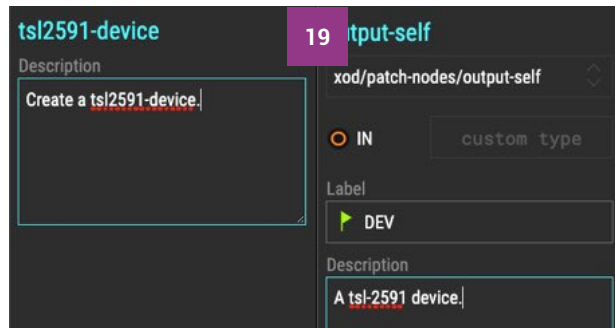
- A** Declare dependencies on the Arduino libraries so that XOD can automatically download and install them.
- B** Include the header files of the Arduino libraries.
- C** Declare a custom type which describes the hardware module.
- D** Create an instance of the custom type.
- E** The evaluate function is called whenever the node requires updating. The `isSettingUp` function returns true on the first transaction. It is used here to ensure that the initialisation code runs once only. The `begin` function of the `Adafruit_TSL2591` class is called to initialise the sensor; if initialisation fails an error is raised.
- F** Finally an instance of type `tsl2591-device` is emitted via the patch terminal node `DEV`. N.B. The custom type takes its name from the patch.

# Documenting your device node & introduction to action nodes



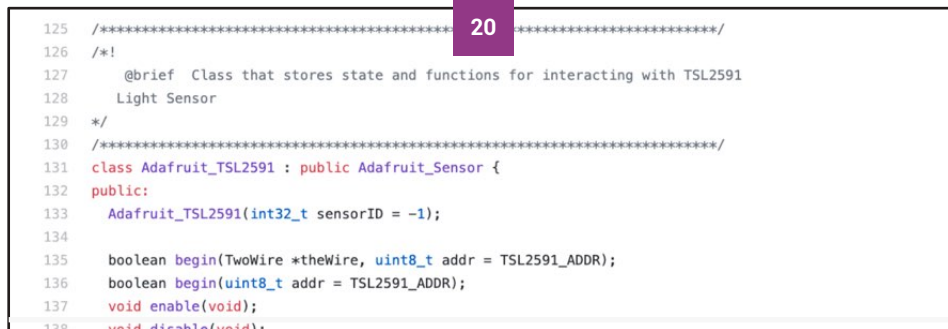
## RETURN TO THE XOD PATCH

You can return to the XOD patch at any time by clicking the back arrow in the top left of the patch.



## DESCRIBE YOUR NODE

Click on an empty space in the XOD patch, then use the description box in the Inspector pane to write a short description of your new node. You can also add descriptions for each node within your patch by clicking on them. This documentation is important as it will help others to understand and use your libraries.



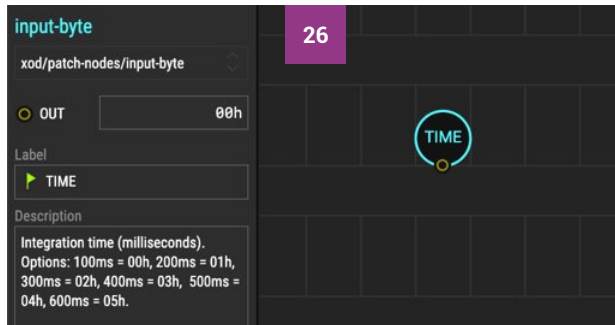
## ACTION NODES

Now that we have a node to represent our device, we also need action nodes to initiate actions or sequences from the Arduino library. In the header file, you can see that the Adafruit\_TSL2591 class has several member functions for configuring and reading data from the sensor. We can make these functions available to XOD by wrapping them inside nodes. As an example we'll use the function to set the integration time (the length of time the sensing element is collecting charge) of the device.



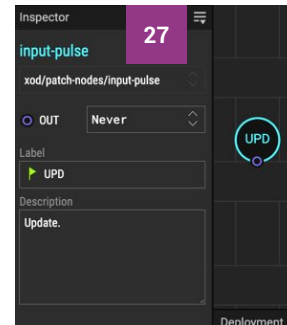






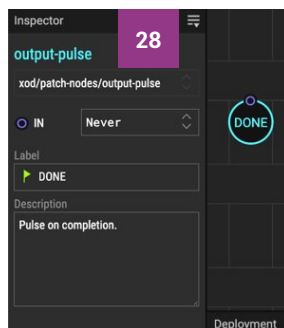
### INPUT-BYTE

Name this node 'TIME' with description 'Integration time (milliseconds). Options: 100ms = 00h, 200ms = 01h, 300ms = 02h, 400ms = 03h, 500ms = 04h, 600ms = 05h'. There's no enum data type in XOD, so we'll use a byte to specify TIME and list the available integration times and their byte values in the description.



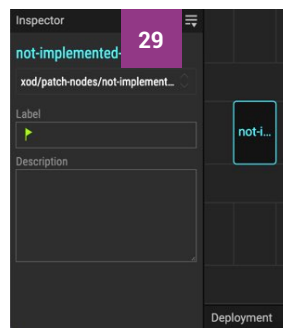
### INPUT-PULSE

Name this node 'UPD' with description 'Update'. Pulses received by UPD will trigger the action of the node.



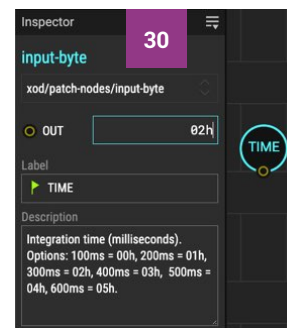
### OUTPUT-PULSE

Name this 'DONE' with description 'Pulse on completion'. This node will output a pulse when the integration time has been set.



### NOT-IMPLEMENTED- IN-XOD

We will use this node to add C++ code linking the XOD patch to the Arduino library.



### DEFAULT VALUES

We can set default values for inputs. E.g. set default integration time to 300ms using 02h in the OUT field of the TIME input.



# The Set-Timing C++ code

```

1 node {
2   void evaluate(Context ctx) { 31
3     // The node responds only if there is an input pulse
4     if (!isInputDirty<input_UPD>(ctx))
5       return;
6
7     // Get a pointer to the `Adafruit_TSL2591` class instance
8     auto sensor = getValue<input_DEV>(ctx);
9     sensor -> setTiming(getValue<input_TIME>(ctx));
10    emitValue<output_DONE>(ctx, 1);
11  }
12 }

```

## REPLACE THE TEMPLATE WITH CODE

Double-click on the **not-implemented-in-xod** node to open the C++ editor. Replace the template with the code below. Read the comments for an explanation of each line.

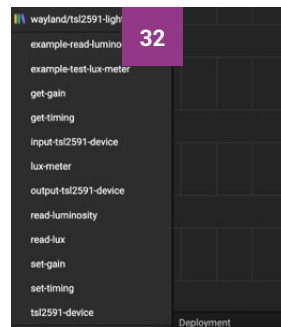
## C++ CODE FOR XOD SET-TIMING NODE

```

node {
  void evaluate(Context ctx) {
    // The node responds only if there is an input pulse
    if (!isInputDirty<input_UPD>(ctx))
      return;

    // Get a pointer to the `Adafruit_TSL2591` class
instance
    auto sensor = getValue<input_DEV>(ctx);
    sensor -> setTiming(getValue<input_TIME>(ctx));
    emitValue<output_DONE>(ctx, 1);
  }
}

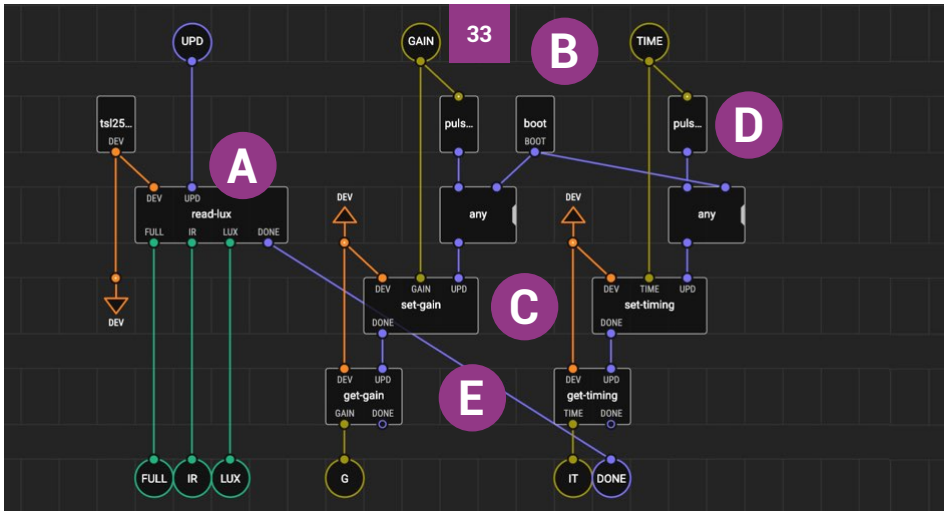
```



## REPEAT FOR EACH FUNCTION

Repeat the process for each of the functions in the Arduino library. Use the **wayland/tsl2591-light-sensor** library as a reference.

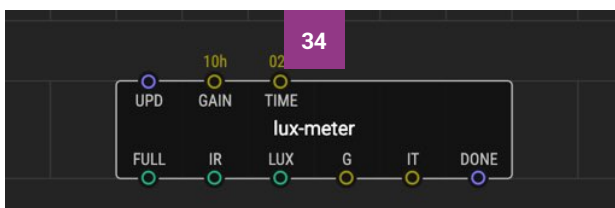
# Creating a Quick-Start node



## CREATE A QUICK-START NODE

Let's simplify use of our library by creating a single node with all the functionality a typical user requires. For the TSL2591 sensor, we will assemble a lux meter.

- A** The **read-lux** action node is triggered by a pulse to UPD and outputs total luminosity (FULL), infrared luminosity (IR) and lux (LUX).
- B** The inputs GAIN and TIME are used to set sensor gain and integration time respectively.
- C** The **set-gain** and **set-timing** action nodes are triggered on the initial boot and also whenever the input values change.
- D** The **pulse-on-change** nodes (**xod/core**) emit a pulse when the values of their inputs change.
- E** The **get-gain** and **get-timing** action nodes report the current sensor gain and integration time respectively.

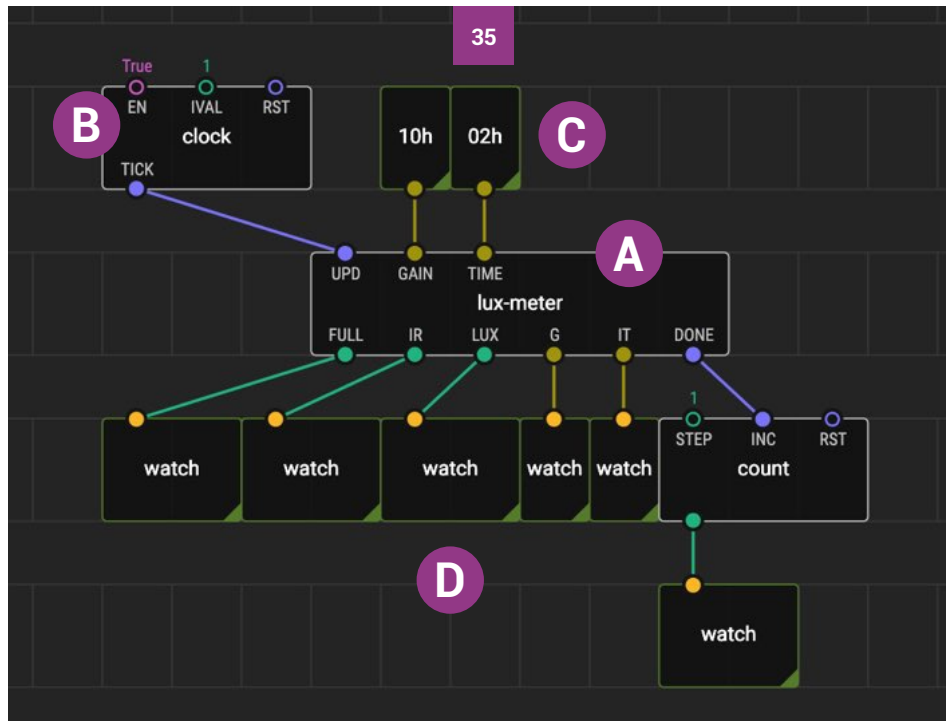


## LUX-METER NODE

The finished lux-meter node will look like this.



# Creating example patches and testing



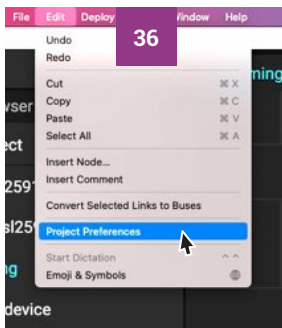
## MAKE AN EXAMPLE PATCH AND TEST YOUR PATCH

Example patches demonstrate how to use your library and are also invaluable for testing. This example patch shows how our newly-created **lux-meter** node can be used.

- A** The **lux-meter** is our quick-start node which encompasses our device node, as well as several action nodes, to take readings from the sensor.
- B** A clock node is used to initiate a reading from the sensor every second.
- C** **Tweak** nodes allow the user to adjust the gain and integration time at runtime.
- D** **Watch** nodes display the values output from the **lux-meter**.

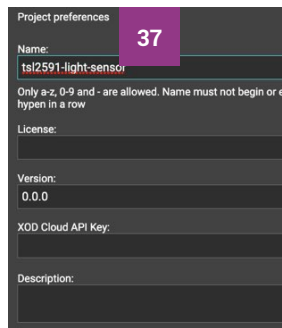
Once finished you should use your example patch to test your nodes. Use 'Upload and Debug' to upload the patch to your Arduino, installing dependencies if you need to. Once running you should see output to all **watch** nodes. Check whether the values being reported by **watch** nodes are sensible, and try adjusting the gain and integration time.

# Publishing your library



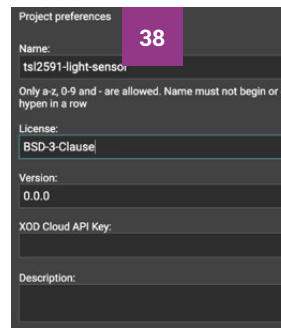
## OPEN PROJECT PREFERENCES

The first step to publish your library is to set the metadata. Go to 'Edit' > 'Project Preferences...' in the menu bar.



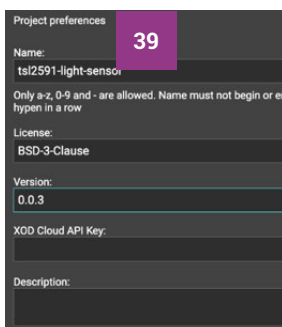
## SET METADATA: NAME

Use this window to set your library's metadata. Under 'Name' add a short, but descriptive name (max 20 characters).



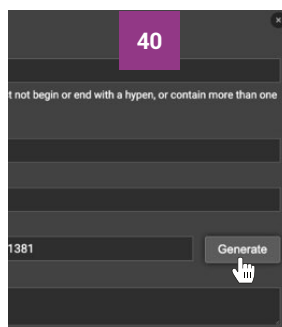
## SET METADATA: LICENCE

Under 'Licence' choose an open source software license (see [www.opensource.org/licenses](http://www.opensource.org/licenses)).



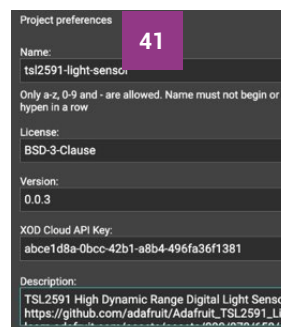
## SET METADATA: VERSION

Under 'Version' set the version number using Semver notation, i.e. major.minor.patch.



## SET METADATA: XOD CLOUD API KEY

Used only for the feeds service provide by XOD Cloud. You can use the 'Generate' button to create an API key.



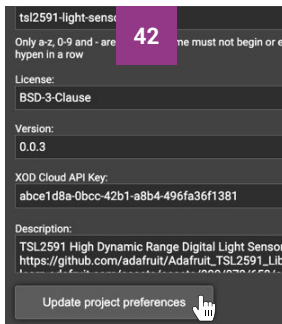
## SET METADATA: DESCRIPTION

Briefly describe the purpose of the library. You can include a link to the Arduino library and the datasheet for the device.



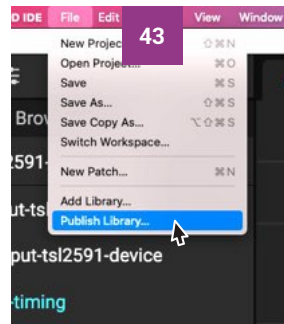


# Publishing your library



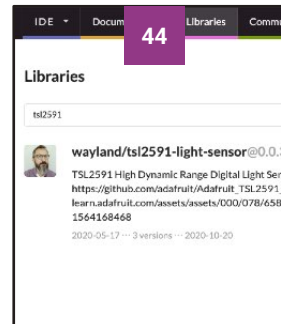
## UPDATE PROJECT PREFERENCES

Click the 'Update Project Preferences' button to save your changes.



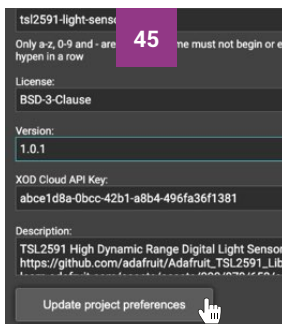
## PUBLISH LIBRARY

When ready to publish, go to 'File' > 'Publish Library...'. A window will summarise the metadata. Click 'Publish' to finalise.



## XOD LIBRARY DATABASE

Your library will now appear in the XOD database ([www.xod.io/libs](http://www.xod.io/libs)), available for other users to download.



## UPDATES

To update your library:

- Open the project.
- Make changes.
- Update metadata.
- Publish again.

## Summary

The process of wrapping class-based Arduino libraries can be summarised as follows:

1. Find Arduino library for device
2. Test Arduino library
3. Familiarise yourself with the class defined by the library
4. Start a new XOD project
5. Create a new device
6. Wrap class member functions in action nodes
7. Create a quick-start node
8. Create one or more example patches
9. Test library
10. Share library with XOD community

# Useful Resources

## XOD Resources

### XOD DOCUMENTATION

XOD has good quality documentation for a range of projects available at [www.xod.io/docs](http://www.xod.io/docs). The following guides are particularly relevant for this tutorial:

- Wrapping class-based Arduino libraries: [www.xod.io/docs/guide/wrapping-arduino-libraries](http://www.xod.io/docs/guide/wrapping-arduino-libraries)
- C++ API: [www.xod.io/docs/reference/node-cpp-api](http://www.xod.io/docs/reference/node-cpp-api)
- Error handling: [www.xod.io/docs/guide/errors](http://www.xod.io/docs/guide/errors)
- Dealing with state: [www.xod.io/docs/guide/cpp-state](http://www.xod.io/docs/guide/cpp-state)
- Dealing with time: [www.xod.io/docs/guide/cpp-time](http://www.xod.io/docs/guide/cpp-time)

### XOD FORUM

XOD has a friendly and helpful community. Don't be afraid to ask for help on the forum at [www.forum.xod.io](http://www.forum.xod.io)

### XOD LIBRARIES

You can learn a lot from looking at existing libraries at [www.xod.io/libs](http://www.xod.io/libs). However, you should be aware that many use an older style of C++ syntax. See [www.xod.io/docs/guide/migrating-to-v035](http://www.xod.io/docs/guide/migrating-to-v035) for more details.

## Arduino Libraries

The following are good locations to search for relevant class-based Arduino libraries:

- Arduino: [www.arduinolibraries.info](http://www.arduinolibraries.info)
- Adafruit: [www.adafruit.com](http://www.adafruit.com)
- Pololu: [www.pololu.com](http://www.pololu.com)
- Sparkfun: [www.sparkfun.com](http://www.sparkfun.com)



# Case Studies

## eCO-SENSE: Soil Sensors Powered by Plant Photosynthesis

This project aims to prototype a low-cost soil sensor powered by biophotovoltaics. The device uses an Arduino Uno, temperature sensor, moisture sensor and gas sensor to take measurements of soil conditions, and adds a bluetooth module to send the data wirelessly to a phone or computer. Working together with Dr Paolo Bombelli from the University of Cambridge, the team aims to use biophotovoltaic cells to power their device, allowing it to be used in-situ and in low-resource environments.

The project used a DHT22 temperature and humidity sensor (similar to the DHT11 sensor on the Grove board), an FC-28 moisture sensor, an SGP30 gas sensor and an nRF8001 bluetooth breakout board. Grove compatible alternatives include the Grove DHT11 or DHT22 sensors (use node `xod/dev/dht2x-hygrometer` and socket D2), the Grove Soil Moisture sensor (use node `xod/common-hardware/analog-sensor` and sockets A0/A2/A6 - beware of clashes), the Grove VOC and eCO2 sensor (convert SparkFun or Adafruit Arduino libraries and use I2C socket, address 58h) and the Grove Blueseed module (use UART socket). Wireless communication is not yet fully supported in XOD, but for a useful tutorial exploring how to send sensor data to your Android phone via bluetooth using Arduino IDE see [www.instructables.com/How-to-Receive-Arduino-Sensor-Data-on-Your-Android](http://www.instructables.com/How-to-Receive-Arduino-Sensor-Data-on-Your-Android).

You can read more about the eCO-SENSE project on their Hackster page:

[www.hackster.io/glen-choa/eco-sense-soil-sensors-powered-by-plant-photosynthesis-be80a2](http://www.hackster.io/glen-choa/eco-sense-soil-sensors-powered-by-plant-photosynthesis-be80a2)

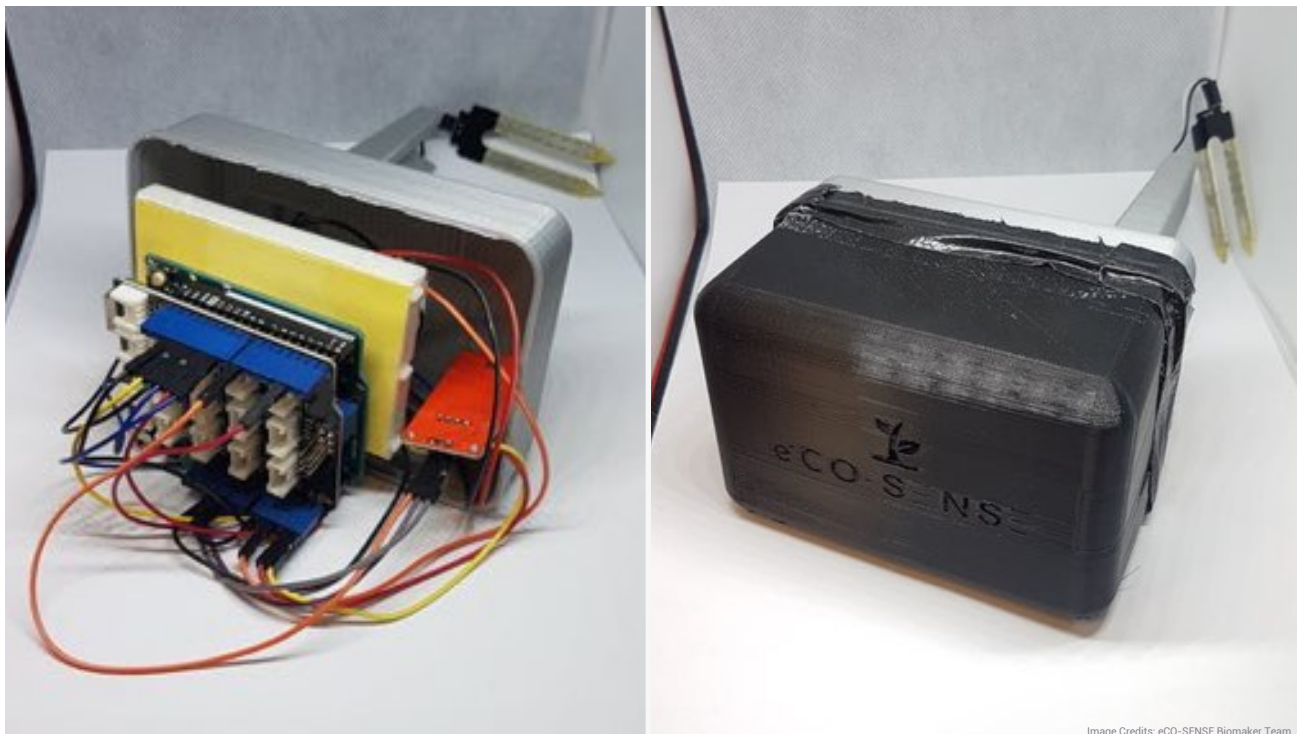


Image Credits: eCO-SENSE Biomaker Team

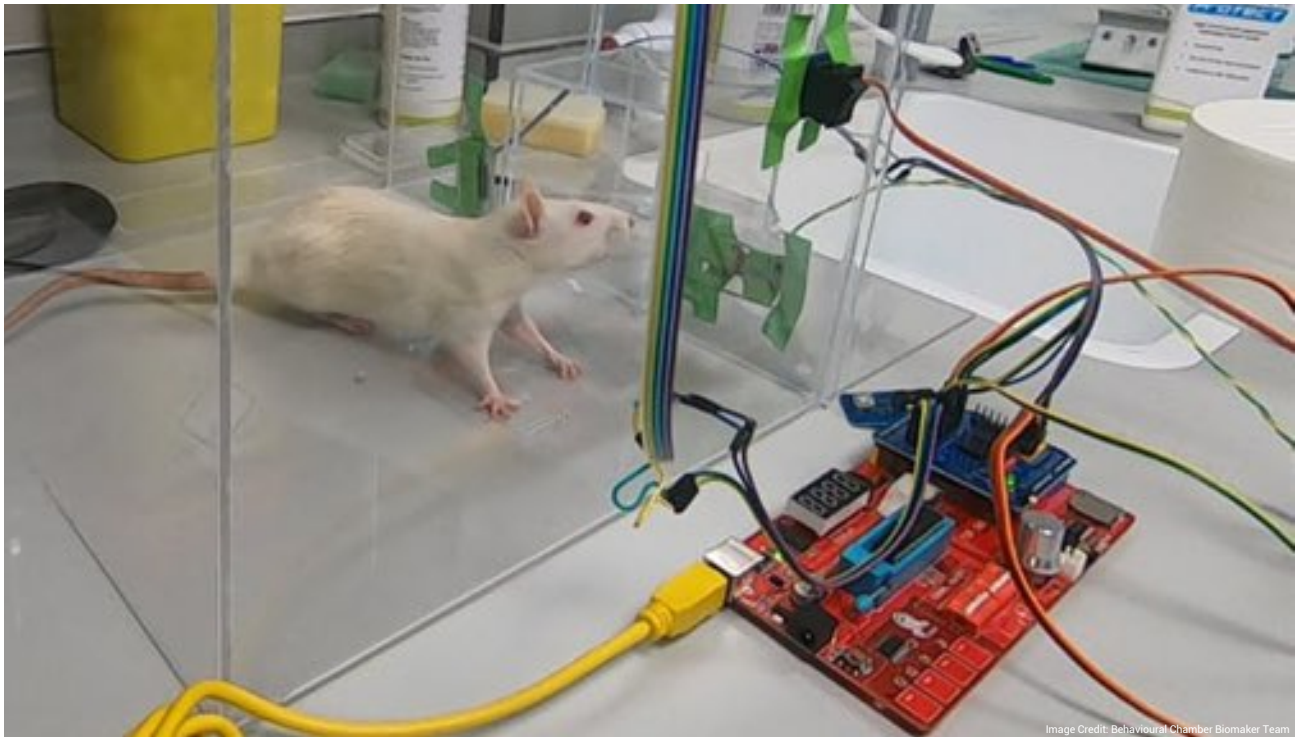


Image Credit: Behavioural Chamber Biomaker Team

## Behavioural Chamber to Evaluate Rodent Forelimb Grasping

This project uses a light emitter and light sensor to monitor when a rodent moves past a certain threshold, and triggers release of a sugar pellet when it does.

The project uses a red laser pointer and GL5528 light sensor to create a trip sensor that notifies the programme when a rodent has crossed a boundary. This then instructs a ULN2003 motor driver to initiate the custom built pellet dispenser. A count of how many times a rodent has completed this task is shown on an LCD screen. Grove compatible alternatives include the Grove Light Sensor (included on the board, use node `xod/common-hardware/analog-sensor` and sockets A0/A2/A6 - beware of clashes), the Grove I2C Motor Driver (use node `gweimer/h-bridge/h-bridge-2dir` and I2C socket, variable address) and the Grove 16 x 2 LCD (use node `xod-dev/text-lcd/text-lcd-i2c-16x2` and I2C socket, address 3Eh).

You can read more about the behavioural chamber project on their Hackster page:

[www.hackster.io/alejandrocarn/a-behavioural-chamber-to-evaluate-rodent-forelimb-grasping-bedb1a](http://www.hackster.io/alejandrocarn/a-behavioural-chamber-to-evaluate-rodent-forelimb-grasping-bedb1a)

# Case Studies

## Camera for Monitoring Plant Pollination Events

This project developed a video and time-lapse monitor to record pollinators interacting with plants. The set up included an environmental sensor to measure temperature, humidity and barometric pressure and wrote this data into the image file names on a microSD card for later analysis.

The project used a Raspberry Pi (see p84), a 160° variable focus camera, a BME280 temperature pressure and humidity sensor and a 128x64 OLED screen. Grove compatible alternatives include the Grove BME280 Barometer sensor (use node [emiliosancheza/bme280-sensor/sensor-bme280](#) and I2C socket, address 76h), the Grove OLED Display 0.96 inch (included on the board, use library [wayland/ssd1306-oled-i2c](#) and I2C socket, address 3Ch) and the Grove Serial Camera Kit with a Grove SD Card Shield. Camera modules are not yet supported in XOD, but information of how to use the Grove Serial Camera is available at [www.wiki.seeedstudio.com/Grove-Serial-Camera\\_Kit](http://www.wiki.seeedstudio.com/Grove-Serial-Camera_Kit). Note that for a high resolution auto-focusing camera, like the one used in this project, Raspberry Pi is a better option than Arduino, as these tasks require high processing power.

You can read more about the plant pollination monitor project on their Hackster page: [www.hackster.io/team-ppi/variable-time-camera-for-monitoring-plant-pollination-events-ad21e7](http://www.hackster.io/team-ppi/variable-time-camera-for-monitoring-plant-pollination-events-ad21e7)

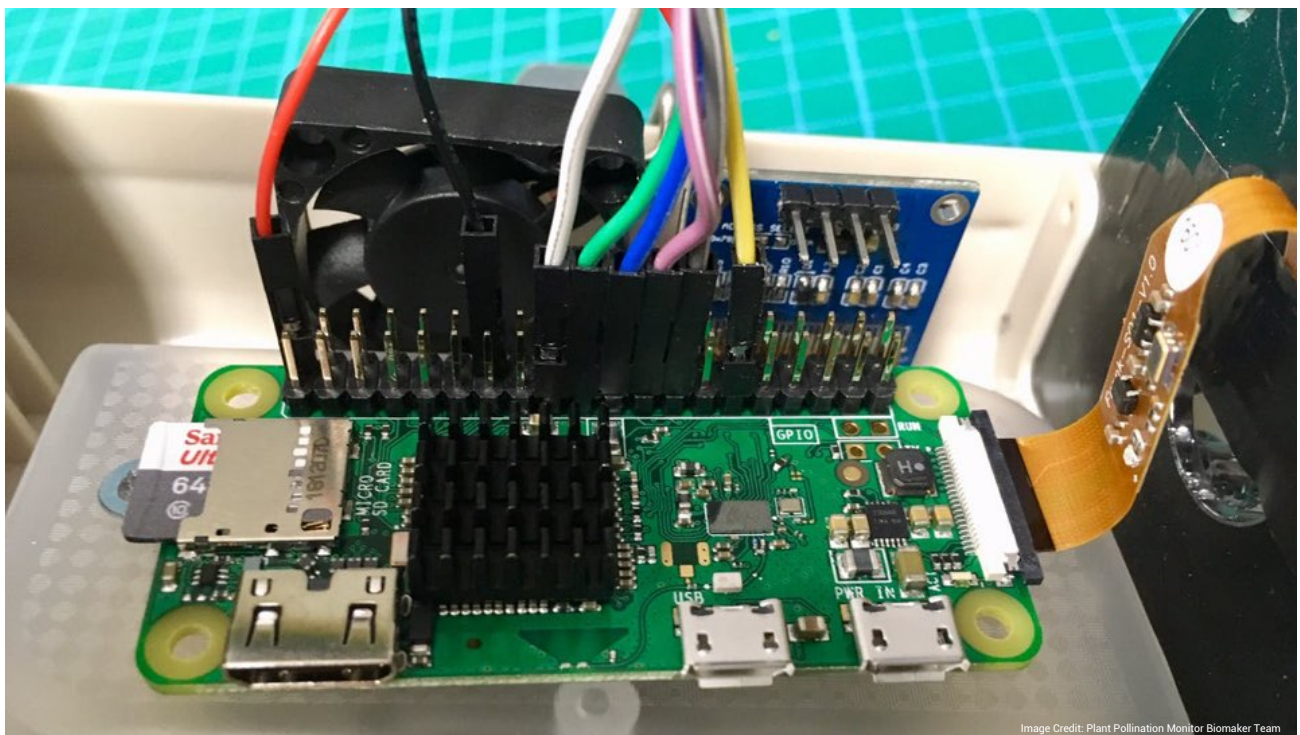


Image Credit: Plant Pollination Monitor Biomaker Team



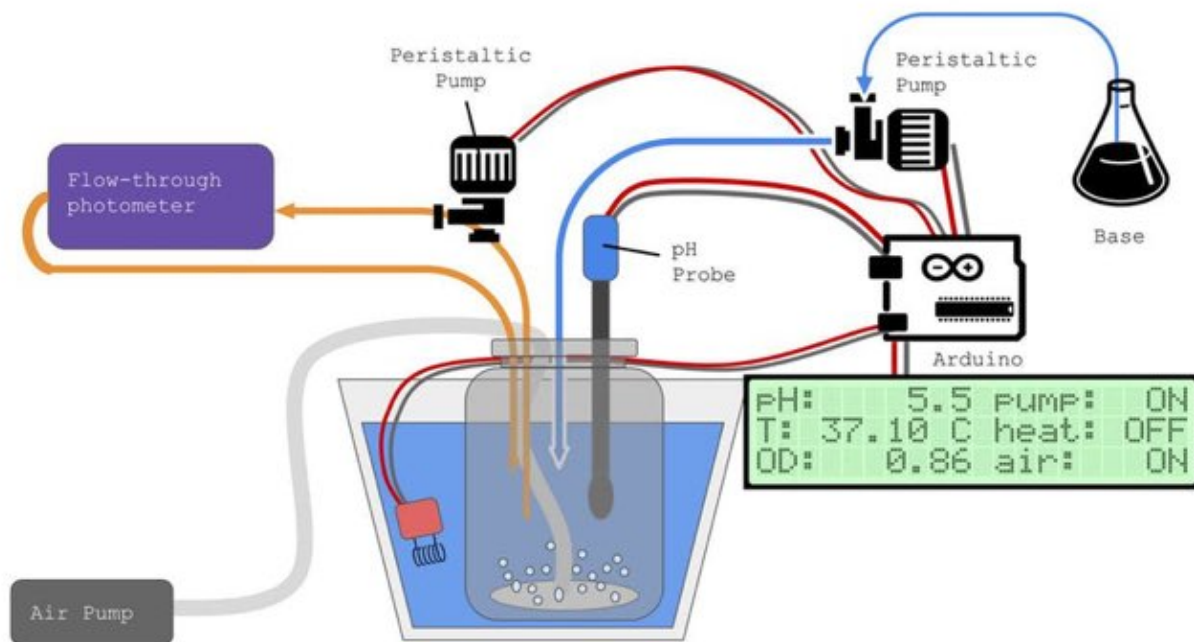


Image Credit: Microbial Bioreactor Biomaker Team

## Open Source Microbial Bioreactor

This project aims to develop an open source bioreactor to optimise yield of enzymes producing recombinant proteins for molecular biology. The bioreactor measures optical density of the culture and monitors and regulates the pH, temperature and aeration.

The project uses an LED and photodiode to measure optical density, a pH probe and peristaltic pump to maintain pH and an LCD screen to display the reactor conditions. The team also aims to add a temperature sensor and heating pad to maintain temperature, and an oxygen sensor and agitation device to maintain aeration. Grove compatible alternatives include the Grove Red LED (included on the board, use node `xod/common-hardware/led` and socket D4), the Grove Light sensor (included on the board, use node `xod/common-hardware/analog-sensor` and sockets A0/A2/A6 - beware of clashes), the Grove pH sensor (use node `xod/common-hardware/analog-sensor` and sockets A0/A2/A6 - beware of clashes), the Grove I2C Motor Driver to drive a peristaltic pump (use node `gweimer/h-bridge/h-bridge-2dir` and I2C socket, variable address) and the Grove 16 x 2 LCD (use node `xod-dev/text-lcd/text-lcd-i2c-16x2` and I2C socket, address 3Eh).

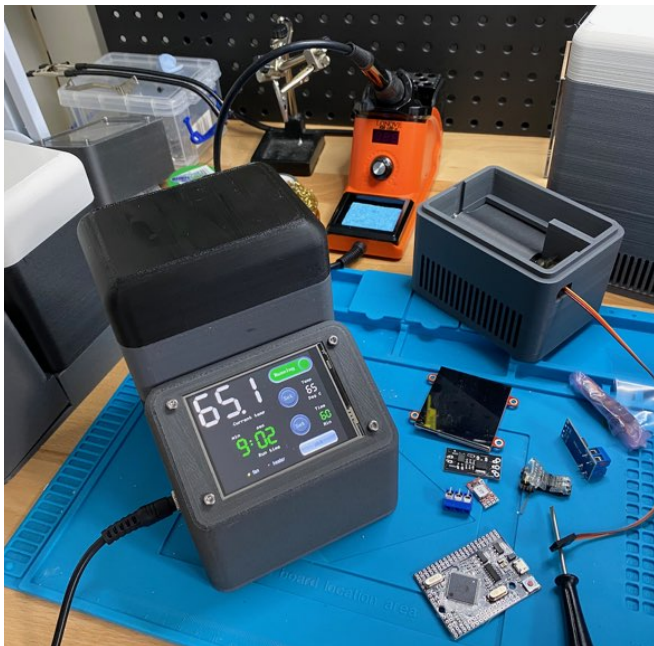
You can read more about the microbial bioreactor project on their Hackster page: [www.hackster.io/open-bioeconomy-lab/microbial-bioreactor-d7f61b](http://www.hackster.io/open-bioeconomy-lab/microbial-bioreactor-d7f61b)

# Example: the AirFlow reactor

## Prototyping of a low cost device for constant temperature incubation of micro reactions, including LAMP molecular diagnostics.

Jim Haseloff, University of Cambridge

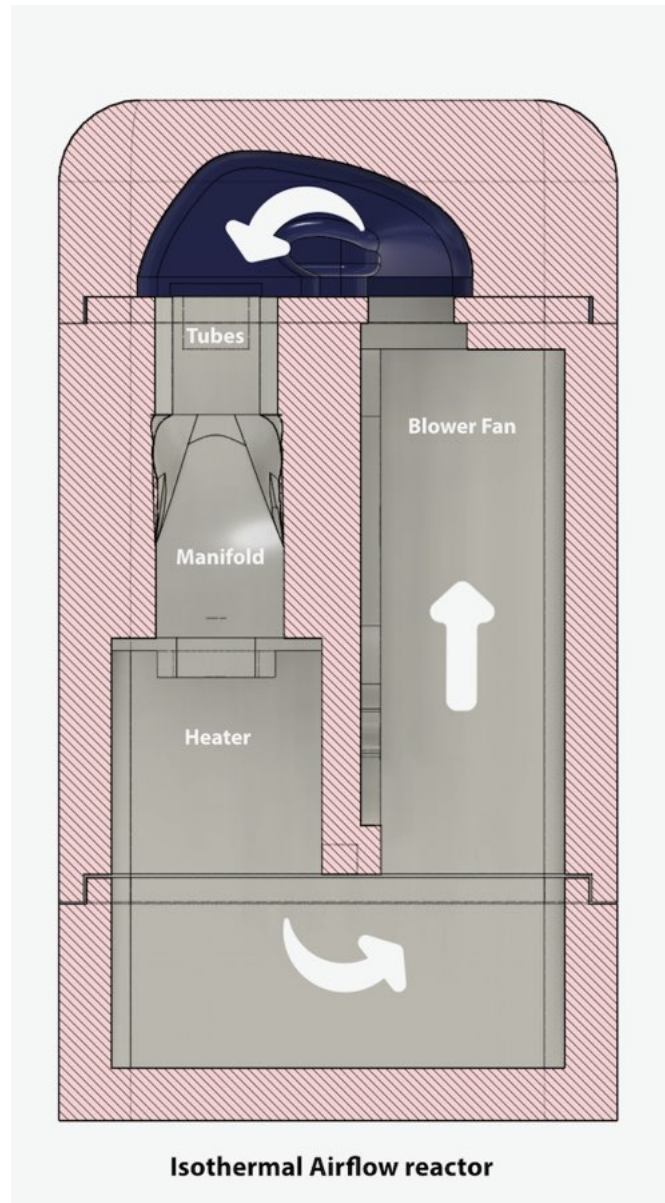
This project describes the design and construction of a low-cost microreactor for temperature control of biological reactions. Such instruments underpin modern molecular diagnostics and synthetic biology approaches that are important for health, agriculture, education and research. However, these instruments are often expensive (\$1000's), and these prices are a barrier to wider adoption and equitable access to advanced biological technologies that can otherwise be implemented cheaply. Early attempts to exploit polymerase chain reaction in the 1990's resulted in a number of designs for thermal cycling systems - this resulted in commercialisation of a number of efficient but expensive designs for the research and medical diagnostics communities. As time has passed, we have seen ever-lowering cost of DNA synthesis, and advent of new molecular tools, which are democratising access to engineering of biological systems. There is now an opportunity to bring these together with the low-cost electronics, optics and manufacturing techniques that have been developed by open source technologists and DIY communities.



### Background

The programme was initiated after the COVID19 lockdown, and was an attempt to explore different approaches to the design of microreactors. The common theme is that heated air flow is used for temperature control. This avoids the need for a machined metal block and complementary heated lid - common but relatively expensive items. A number of early successful designs for PCR machines (e.g. Corbett Rotorgene, Roche Lightcycler) used directed flow of hot and cool air, but the

approach has fallen away, as Peltier device heated/cooled block-based designs have proliferated. It may be time to re-visit this, armed with new advances in electronic control and 3D printing technologies...and with modest intermediate goals, such as low-cost incubators for isothermal diagnostics, like LAMP assays (used for viral diagnosis).



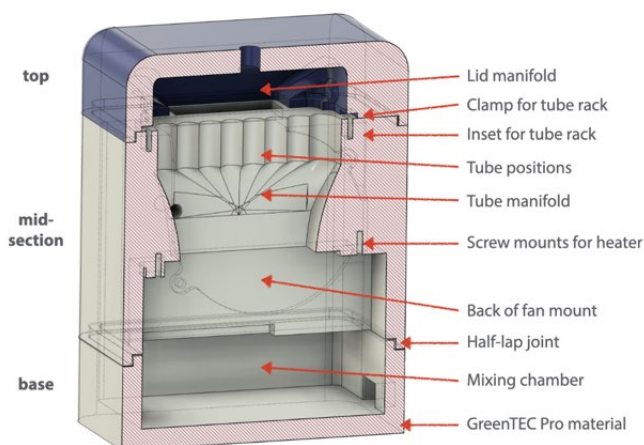
### Specifications

Earlier prototyping experiments have led to the design of a programmable thermal reactor that allows microtube reactions to be incubated in a constant temperature air flow. The objective was to build a device that included:

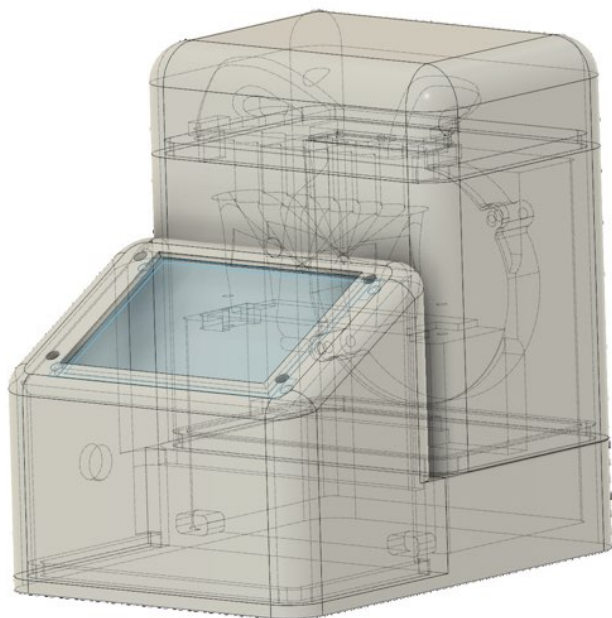
- Space for at least one 8-microtube strip that takes up a volume of approximately 80x30x10mm - with allowance for possible future integration of optical sensors or cheap plastic fibre optics.
- An off-the-shelf PTC resistive element as heat source.
- Unidirectional fan-forced air flow through the device, using

a low-cost computer blower fan.

- Minimal volume of air for recirculation, to improve response to heating (or cooling).
- An accurate thermal control system using low-cost Arduino electronics and sensors.
- Touchscreen microcontroller interface that allowed easy use of the programmable device.
- Attempt to minimise costs of construction and use components that are globally accessible.
- The use of 3D printing, no-code programming, commodity electronics and open source documentation to allow free sharing and modifications of the design.



The project aimed to explore different components, designs and practical assembly of low-cost microtube incubators - which would make accessible a new generation of isothermal reactions - for home testing, field applications and international educational efforts.



#### Instrument design

The core of the instrument is a rack for an 8-microtube strip inside a modular 3D printed set of blocks that create a physical loop for air flow - with temperature control by a computer controlled heater and forced air flow by a blower fan. The device consists of sections that slot together with half-lap joints defined in the 3D print files. Exterior walls are 8mm thick, with

3.75 mm flanges at the overlap between section, allowing 0.5 mm gap between half-lap joint pieces, which are 5mm deep.



This allows sufficient clearance to avoid problems with fitting of curved pieces, and allows the sections to be simply stacked on top of each other to create a fairly stable and air tight joint. This is very useful during prototyping, but might be replaced by something more permanent in a finalised design.

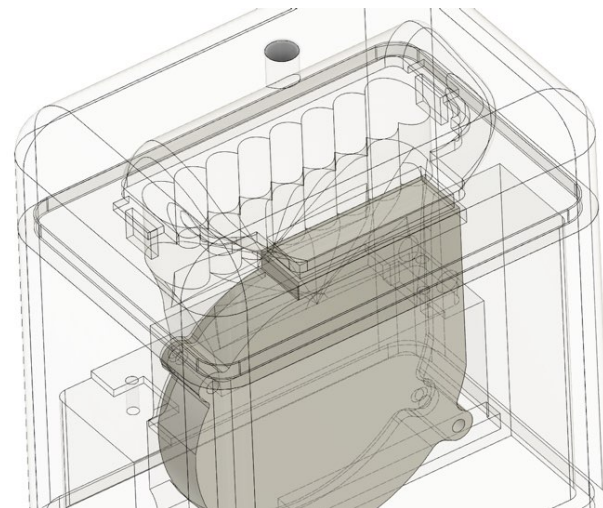
The sections consist of (i) a base plate with an extended front deck for the electronics, (ii) a mid-vessel sections sits above this and contains the heater and fan, and support for the microtube rack. (iii) A custom lid sits above this, and contains streamlined venting to direct forced air flow between the fan and heater compartments. (iv) A front console sits on the front part of the base section, This provides a support for the integrated touchscreen and a housing for the instrument electronics.

The AirFlow parts can be printed in a variety of different materials, but care must be taken if the reactor is to be used at temperatures over 70°C, where commonly used materials such as PLA will start to soften and deform, and may even melt. I have settled on Extruder GreenTEC Pro filament, which is relatively resistant to heat up to 160°C (VICAT softening temperature: at which a specimen is penetrated to a depth of 1 mm by a flat-ended needle with a 1 mm<sup>2</sup> circular or square cross-section - but I try not to stray above 100°C). GreenTEC Pro is also derived from renewable raw materials and is biodegradable. The material is very easy to print with and produced a matt finish which is easy to work with tools or smooth with abrasive pads after printing.

The components are relatively large and can take over a day to print. Print failures were minimised by the use of a print bed adhesive. I have found that Dimafix is an excellent adhesive for use with GreenTEC Pro filament on the heated glass bed found on the Ultimaker S3. The build is released after printing by cooling or refrigerating the plate. Printing is generally trouble-free. Also, GreenTEC Pro Carbon is available for stronger carbon-fibre-infused assemblies, using a CC Red 0.6 print core with the Ultimaker S3.



# AirFlow reactor prototype

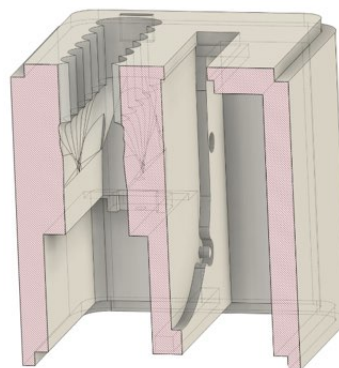


The fan is positioned on the side of an internal wall of the vessel to extract air from that side and to push this into the top chamber. A vent is moulded into the top surface of the vessel section, and the exit port of the fan slots into this. Note: some fans are manufactured with overheat sensors that cause the fan to slow at elevated temperatures - generally  $>70^{\circ}\text{C}$ . The most expensive fans are not always the best. Check what maximum temperature the fan is rated for: different fans bearings can affect this. Running the fan over temperature may shorten its lifespan, but the devices are fairly robust. I have run them in PCR setups at  $95^{\circ}\text{C}$ - $100^{\circ}\text{C}$  without major problems.

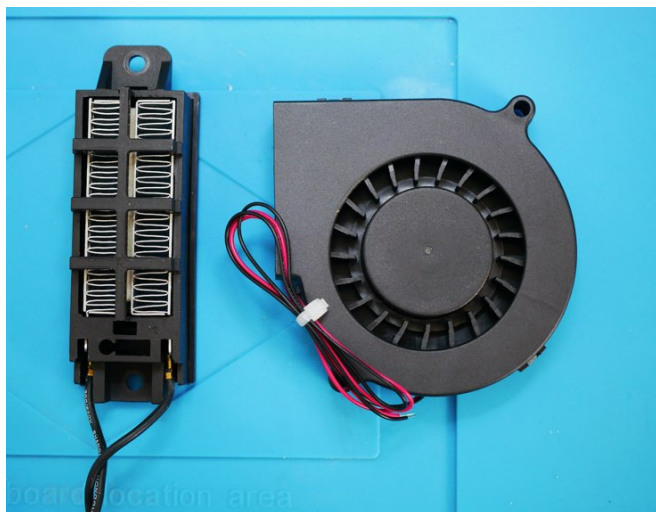
The 3D print designs are available as STL files. The 3D designs include insets and mounting points for the various fixtures - for heater, fan, sensor board, touchscreen, etc. The assemblies were designed using Autodesk Fusion 360, and Fusion 360 files are also available.

## Heater and fan

The two major active components in the device are (i) ARX CeraDyna FW1275-A1041C 12V centrifugal Fan 75 x 75 x 15mm, rated at  $16.58\text{m}^3/\text{h}$ , and (ii) a 50W, 12V PTC heater element that is generally used as a component for heating car windscreens. This simple device consists of a heat resistant plastic housing and PTC heating elements attached to metal radiator fins. The heaters and fans cost around £5-£10 each retail in the UK, and about half that when delivered when from suppliers in China.



In addition, a custom inset has been programmed into the 3D print - positioned as a retainer for for the fan so that it can be clipped into place, along with the slot for the exit port.



The fan can be permanently fixed in place by application of blobs of UV cured glue (liquid plastic), which sets after a few seconds of illumination. This quick-setting fixing agent is generally useful as a way of immobilising printed parts, components and stray leads.

The PTC heater assembly is mounted from the lower side of the vessel mid-section. It slots into a complementary notch engineered into the base. The 3D printed insets for both the fan and heater are customised for the type of component, so may need to be customised for different models.



The PTC heater is fixed in the slot with two self-tapping screw. This fixes the component firmly underneath the manifold.

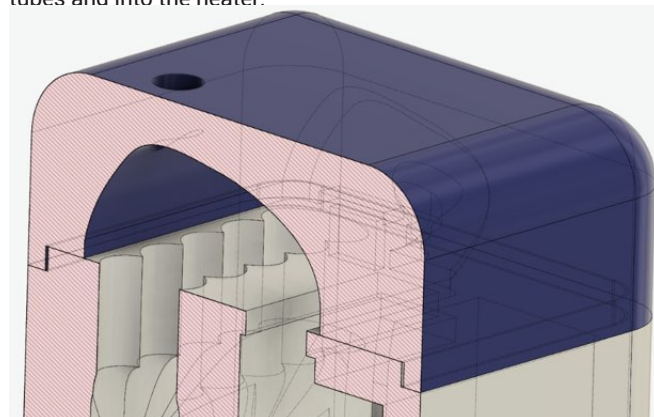


The mounted fan and heater are fixed in the vessel, with leads ready for connection to the electronics in the base section.

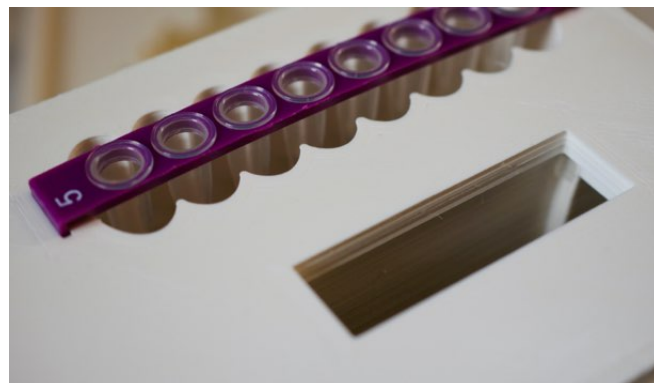


### Manifold and tube rack

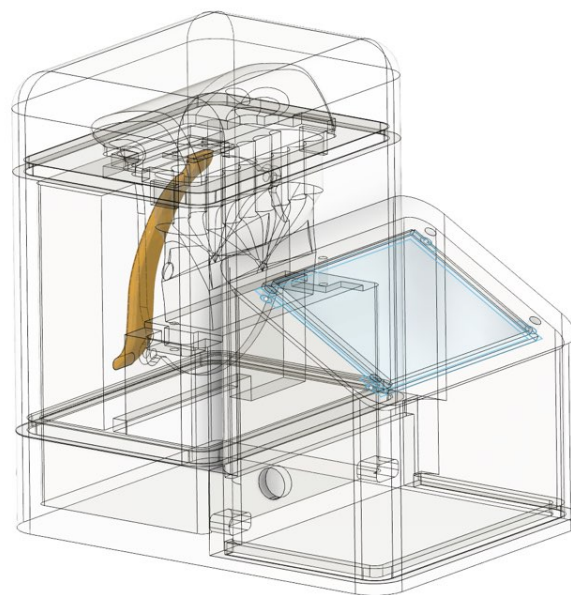
The manifold is incorporated by printing streamlined channels that sit under the microtube rack. The channels provide a funnel for air flow from the top section of the reactor, past the reaction tubes and into the heater.



8-microtube strips fit into the manifold with space around the the sample tubes for airflow.



An accurate digital sensor (MCP9808) is mounted in the top surface of the vessel, under the lid and exposed to the flow of air running towards the sample tubes and manifold. The sensor is mounted on a small circuit board at the top of the reactor, and needs to be connected to the control electronics in the base of the device. A channel is laid down in the 3D printed vessel and provides a conduit for a cable connecting the sensor and control electronics.



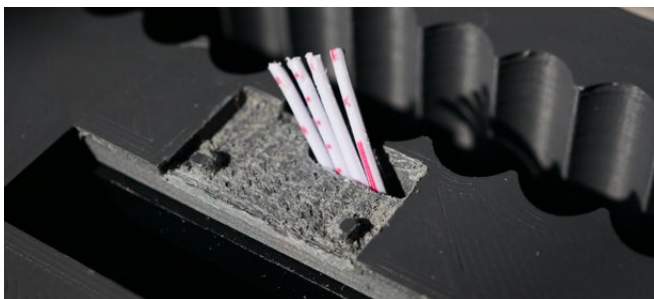


# AirFlow reactor prototype

The sensor is an I2C device and is connected using a 4 wire ribbon cable. A double-headed connector wire can be used. On one end of the cable can be connected to the control electronics. The other end is cut off. The cut end of the flat cable is fed into the internal channel in the 3D printed vessel part.



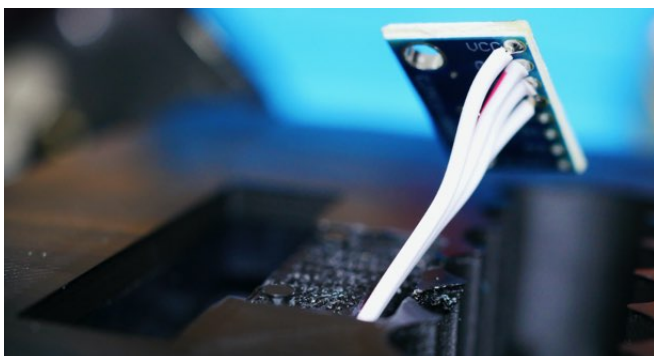
The cable should be pushed through until it emerges from the other end of the printed channel.



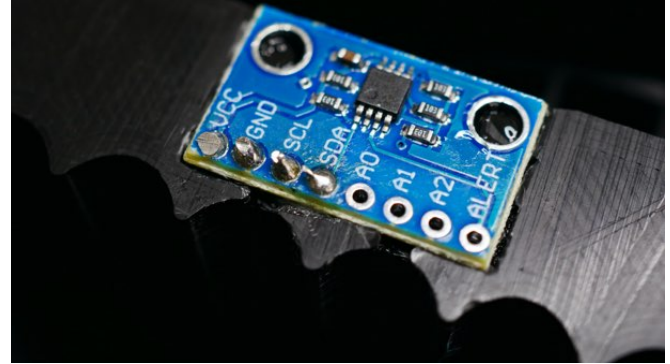
The flat cable wires should then be separated, a few millimetres of insulation stripped off, and the wires tinned with a small amount of solder.



The leads are soldered to the MCP9808, taking note of the 4 connections to the VCC (5V), GND (ground), SCL (clock) and SDA (data) terminals. After soldering, any protruding leads should be snipped off.



The top surface of the vessel is printed with an inset, so that the board can be pushed downwards to sit flush with the surface. Ensure that the solder joints are solid. It can then be permanently glued in place.

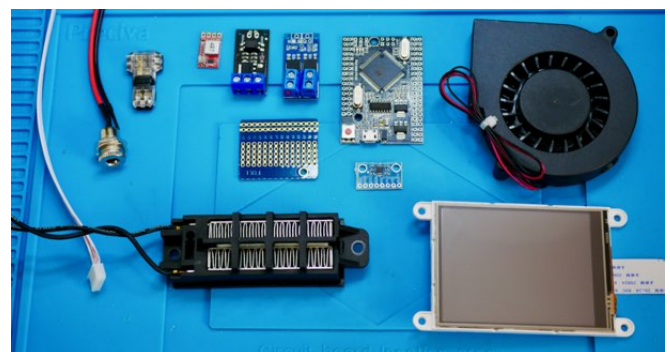


The sensor sits adjacent to the sample tubes and provides an accurate measurement of air temperature. Technical details about the MCP9808 sensor and its use with the XOD no-code programming environment can be found [here](#).



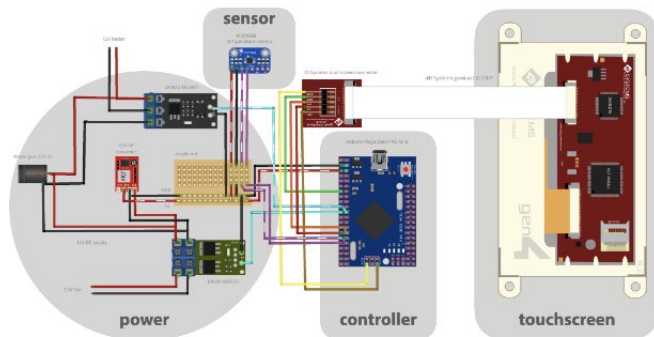
## Control electronics

The basic hardware: heater, fan and temperature sensor need to be hooked up to an electronic control system. Based on earlier experiments (see [hackster.io/jim-haseloff](#)), I decided to use an Arduino-compatible Mega 2650 PRO mini microcontroller board with a 4D Systems Gen4 touchscreen for interactive display and controls. These allow the use of simple and fast no-code programming tools (XOD and 4D Workshop) to configure the control routines and display. Further the Mega 2650 PRO board is both compact and relatively powerful. Unlike Arduino UNO compatible boards (with single hardware serial port), it provides multiple hardware serial ports, which allows easy use of dual ports for programming and screen communication. Hardware serial port connection speeds are much faster than software-defined serial port communication.

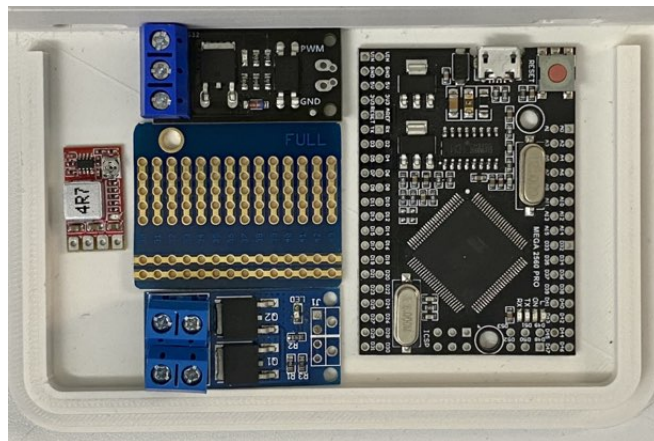




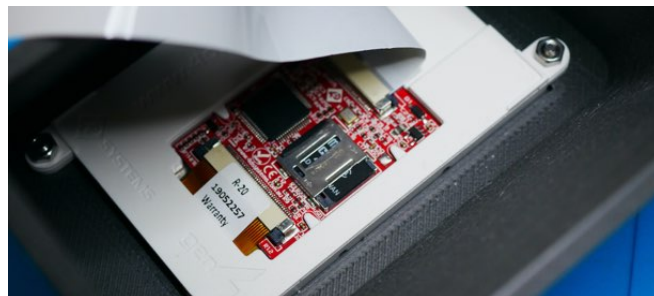
In addition to the microcontroller board and display, the control electronics include a step-down voltage controller to provide a



regulated 5V supply from 12V, and two MOSFET (metal-oxide semiconductor field-effect transistor) solid-state relays for microcontroller regulated, independent switching of power for the fan and heater. The compact size of the components allows the electronics to fit within the tray underneath the 3D printed console at the front of the instrument. In addition, a small piece of stripboard is used to distribute signals from the MCP9808 sensor. A diagram of the full circuit layout can be downloaded (below) either as an image or Fritzing file.

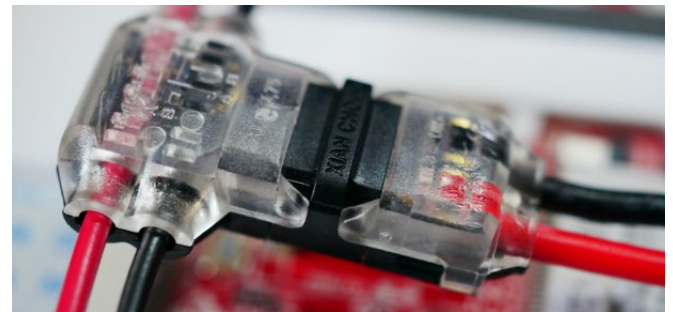


First, assemble the 4D Systems Gen4 uLCD 32DT screen to the back of the 3D printed console. I have variously printed the console section with GreenTEC Pro. If you're expecting rough handling, you can use GreenTEC Pro Carbon filament for a considerably stronger print for this relatively thin-walled console section. The 3D print design includes a cutout and inset to mount the screen neatly, and well as mounting holes for stainless steel M3x10mm bolts. The ribbon cable connector and µSD card can be added for programming the screen. More info at: <https://www.hackster.io/jim-haseloff/biomaker-tutorial-4-programming-the-4ds-touchscreen-3b2006>.

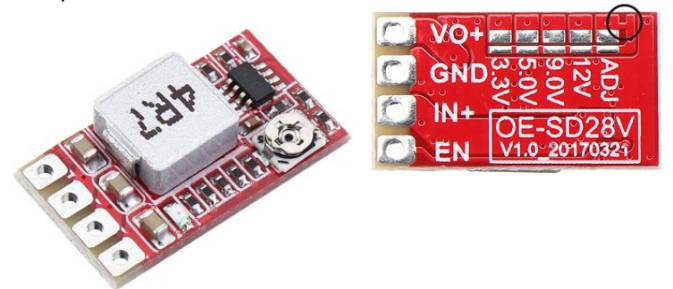


The device has 12V DC heater which is rated at 50 Watts. This is by far the major draw on the power supply, which must be rated to accommodate this. The external 12V DC supply should be rated for at least 6 Amps (72 Watts). Suitable power supplies

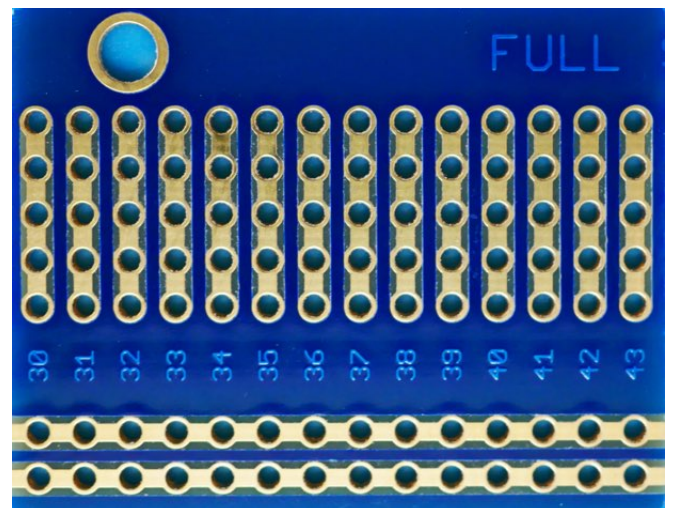
(also used for laptops, CCTV systems, etc.) are readily available for around £10-15 retail price. These power supplies will generally have a 2.1 mm diameter coaxial plug connector, with centre positive polarity (check specifications before purchase). The external power supply will plug into the AirFlow device via a chassis-mounted 5.5 x 2.1 mm DC power socket rated for 10 Amps.



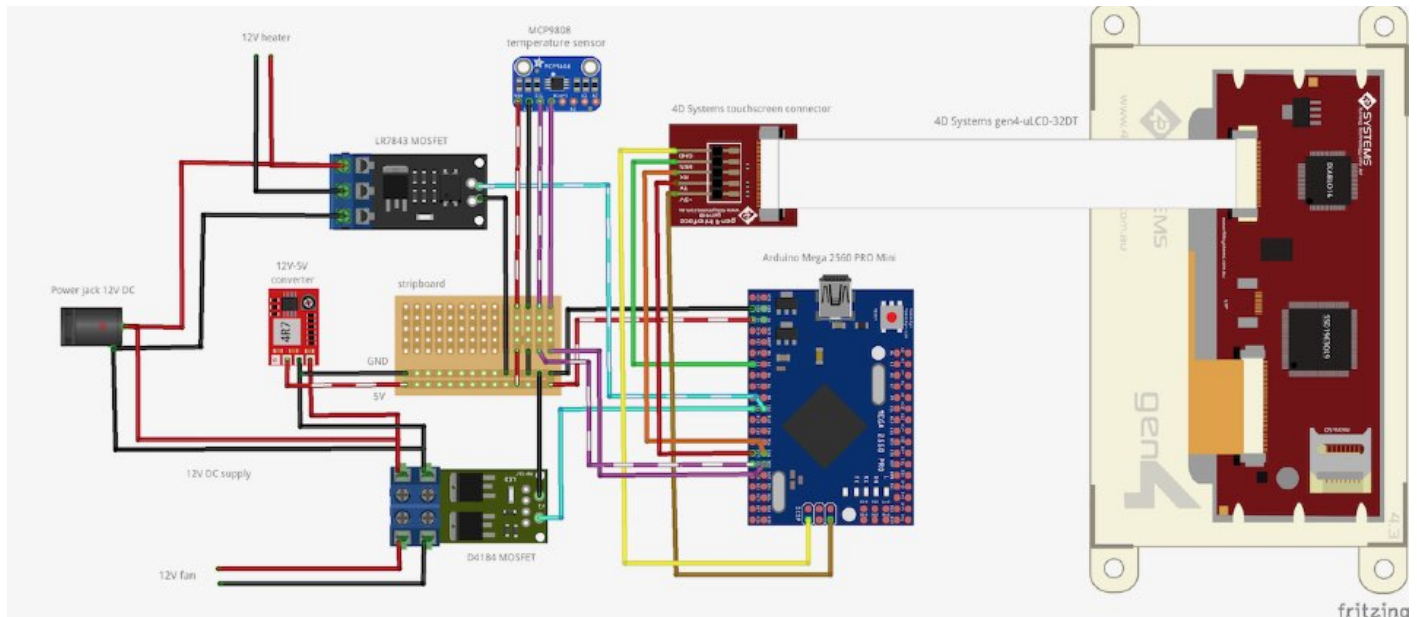
Appropriate gauge wiring should be used for the lines handling the 12V DC supply for the heater. I used 20AWG wire for the main 12V DC lines. Short leads were soldered to the chassis-mounted power supply socket. Bare joints were covered in heat-shrink tubing. Red and black wires were used for positive and negative polarities, respectively. The 12V supply is fed into two branches, one containing a MOSFET switch for the heater circuit, the other containing a MOSFET switch for the fan and a 12V-5V converter circuit board. The two branches were supplied using a T-Tap 2-way quick splice wire connector, where the appropriate 20AWG wire leads were connected by simply clamping in the connector with pliers.



The 12V power supply wires should be connected to: (i) the input screw terminals for the LR7843 MOSFET, which is used to switch power to the PTC heater, and (ii) the input screw terminals for the D4184 MOSFET which switches power to the fan. The terminals are also used as a junction to feed 12V DC to a mini DC-DC Buck stepdown converter (OE-SD28V) to provide a 5V regulated output.

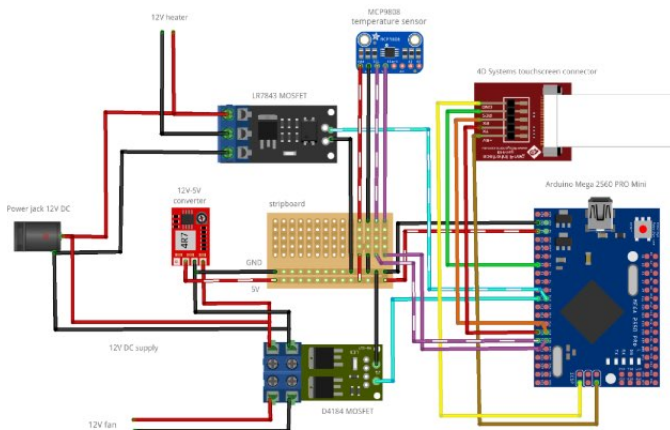


# AirFlow reactor prototype



The circuit board has positive polarity pads for input (IN+) and output (VO+) voltages and a common ground pad (GND). The enable (EN) pad can be used as a switch for the output. The reverse side of the circuit board contains a trace (circled) that can be cut to set the voltage conversion to a set value (rather than adjustable via the onboard potentiometer) and can be set to 3.3V, 5.0V, 9.0V, 12V by bridging solder pads. The circuit requires a 5V supply, so the trace should be carefully cut and the 5V setting chosen. It is a good idea to check the output voltage before wiring it into a circuit. The OE-SD28V circuit board is small component that can be wired up, then enclosed in heat-shrink wrap to make an in-line device.

Wait before wiring the leads to the screw terminals, to allow easier assembly of the rest of the circuit.

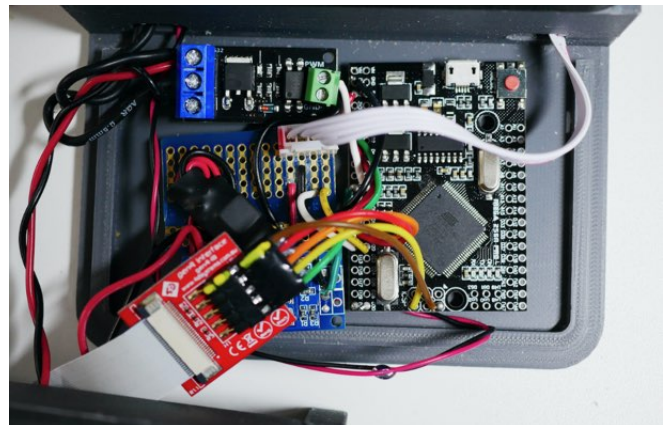


The remaining circuit can be soldered together, to link the microcontroller board, sensor, touchscreen display and MOSFET switches via a small stripboard.

1. cut and trim a small piece of stripboard to suit (see pattern of traces above)
2. Solder a 4 pin plug socket that will accept the signals from the MCP9808 I2C temperature sensor.
3. Connect the 5V, GND, SDA and SCL terminals on the Mega

2560 PRO Mini microcontroller (MCU) to the stripboard.

4. Connect MCU terminals D20 and D21 to the MCP9808 SDA and SCL on the stripboard. Connect the MCP9808 5V and GND lines on the stripboard.
5. Connect MCU terminals D10 and D11 to the two MOSFET modules. Run the GND signals to the GND rail on the stripboard.
6. Connect the D3, D18, D19, 5V and GND signals to the connector for the LCD touchscreen. Leads with a 5 position in-line socket can be soldered to the terminals to allow reusable connection to the 4D Systems touchscreen connector board.
7. The boards should be laid out in the intended positions. Hookup wire can be cut to size, ends stripped and soldered in place. Solid-core wire can be used to create a more rigid framework to keep the boards connected.



After wiring the MCU board and peripheral components, the MCU can be programmed from XOD through its USB port, and the 4D Systems display programmed from 4D Systems Workshop4 via the serial pins on the connector board. (see below for details) The fan, heater and sensor leads pass through the mid-section vessel via ports in the 3D printed casing - and are lead out to the front space under the casing. It is necessary to block any



openings between the reactor vessel and the console space that houses the electronics. I have found that Blu-tack acts as an excellent putty-like stopper that is easily removed and can be reused multiple times during prototyping and debugging.

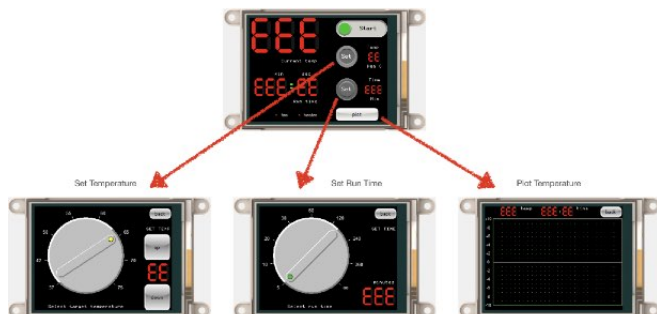
The power supply routing should be attached to the relevant MOSFET screw terminals.

1. Connect the fan and heater leads to the relevant MOSFET screw terminals.
2. Connect the MCP9808 sensor cable to the socket installed on the stripboard.
3. Connect the display to the MCU via the 4D Systems connector board.

The MCU control system and touchscreen user interface were laid out using two no-code graphical programming systems.

**First**, the user interface was composed in 4D Systems Workshop4. This software package is provided free by the manufacturer of the touchscreen. Individual screens are composed by adding widgets in a WYSIWYG editor, and adjusting their properties by setting parameters. Screen widgets can either display status, or be used to set particular values via a range of interactive controllers. Prototype code is then loaded onto the target screen, which results in a defined set of software modules that a microcontroller can communicate with, through a serial port.

**Second**, XOD has good support for 4D Systems screens, and allows the rapid prototyping of control systems.



**Interface design**

The objective was to produce a simple-to-use graphical interface for setting isothermal incubations. The prototype interface consisted of four screens with (i) a top-level status screen that included calls to three subsidiary screens that allowed (ii) setting of the target temperature, (iii) time of incubation, and (iv) screen plot of the vessel temperature over time, displaying +/- 10°C around the set point.

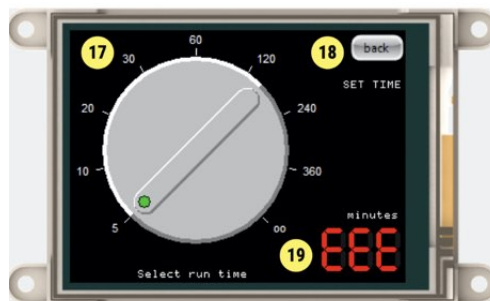


1. ILedDigits0
2. ILedDigits3
3. ILedDigits2
4. LED0
5. LED1
6. ISwitch0
7. IButtonD0
8. IButton D3
9. Winbutton3
10. ILedDigits1
11. ILedDigits5

The top screen has widgets to display: current vessel temperature (°C), current run time (min), temperature set point (°C), set time (min), fan activity (on/off), heater activity (on/off), and buttons to start/stop the run, and to navigate to the three other control screens.



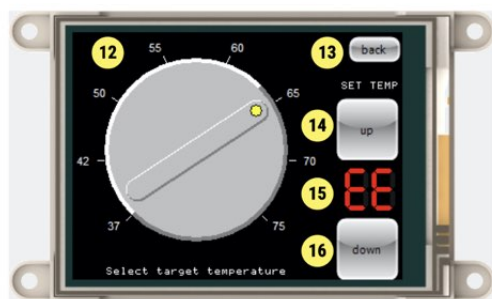
The time of a run can be set from this screen, with a 9-position rotary switch that is touch activated. A series of set times is provided, ranging from 5 min to unlimited. The actual times can be easily modified in software, or interactive controls added. (Most LAMP diagnostic reactions run for 60 min).



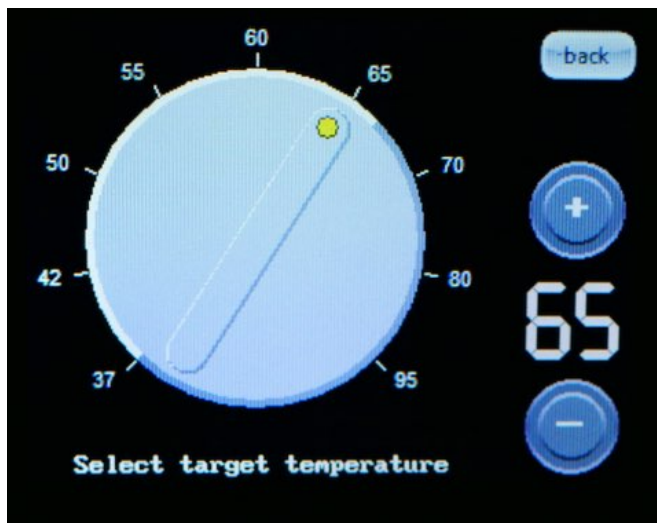
17. Rotaryswitch0
18. Winbutton17
19. ILedDigits11



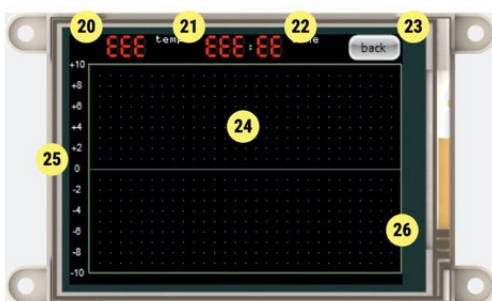
# AirFlow reactor prototype



- 12. Rotaryswitch1
- 13. Winbutton0
- 14. Winbutton2
- 15. ILedDigits4
- 16. Winbutton4



The set point for heating of the vessel can be set from this screen (above). A 9-position switch allows choice of commonly used incubation temperatures up to 95°C. Additional buttons are used for fine adjustments of the target temperature. (Note: it is necessary to use a heat-resistant thermoplastic filament such as GreenTEC Pro if temperatures above 70-100°C are used).



- 20. ILedDigits7
- 21. ILedDigits8
- 22. ILedDigits9
- 23. Winbutton1
- 24. Scope0
- 25. Scale0
- 26. Border0

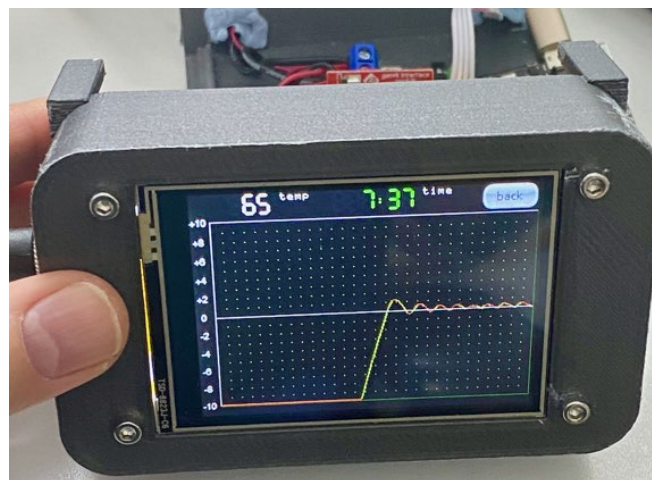
The temperatures during a run are plotted on screen on a moving display. The current temperature is shown on the right-most side of the plot, as well as a numerical value at the top of the screen. The Y-axis shows temperatures within +/- 10°C of the current setpoint. The X-axis represents the elapsed time, with 20 sec between divisions. The total elapsed time is shown as numerical value at the top of the screen. The yellow plot displays the activation of the heater in the control circuit, with 2-step values, high corresponding to on, low to off. In this run, the heat was continuously on, until the setpoint was approached - this was followed by a small amount of hysteresis and oscillation (<1°C) with gradual settling. The degree of overshoot is dependent on the settings of the control software (using a XOD-encoded PID

routine).

All of the subsidiary screens have a 'back' button that triggers return to the top display.



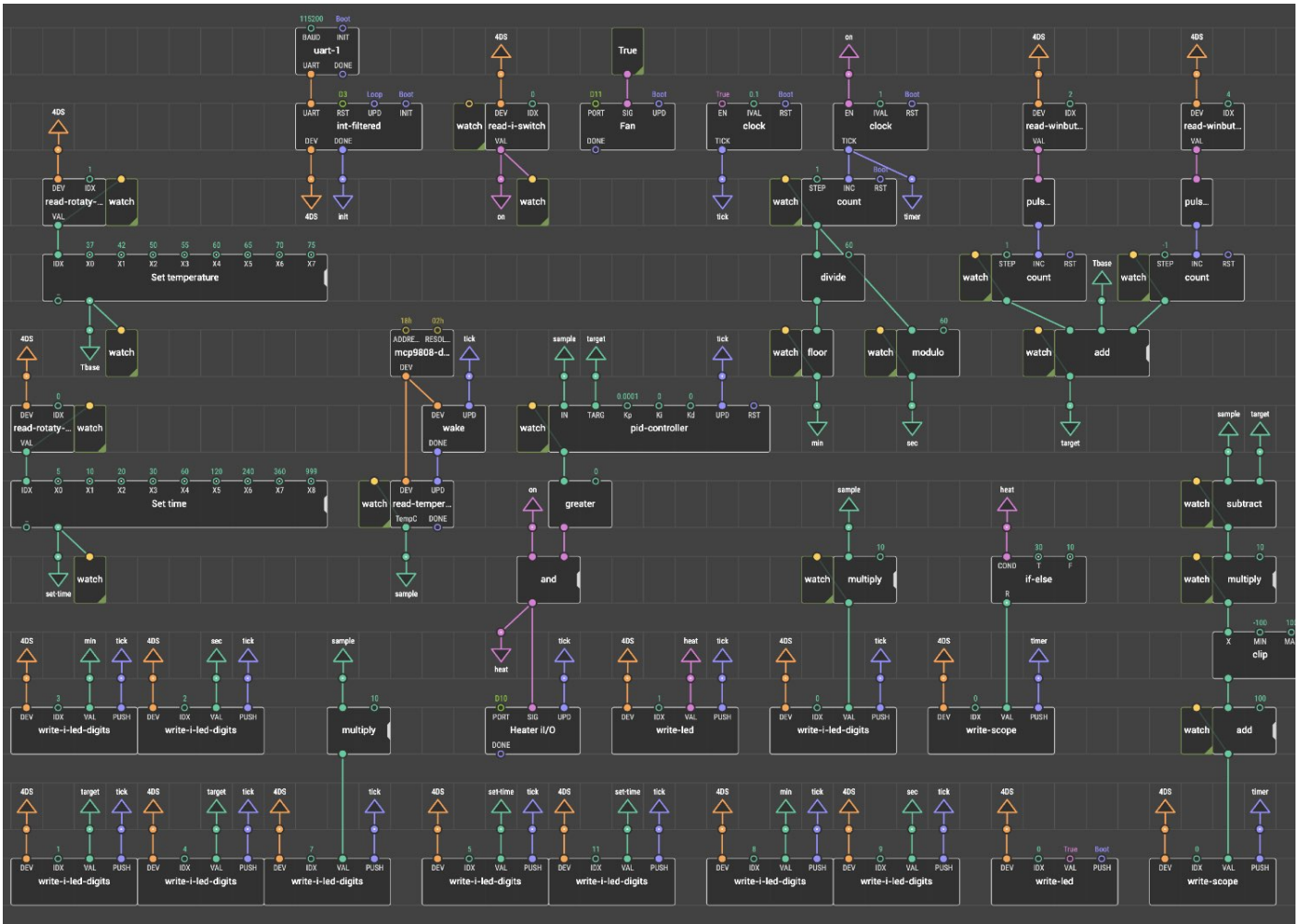
The Workshop4 code that is required to program the 4D Systems touchscreen is available. Individual widgets are addressable from XOD and communication between the two no-code systems is required to build an effective instrument.



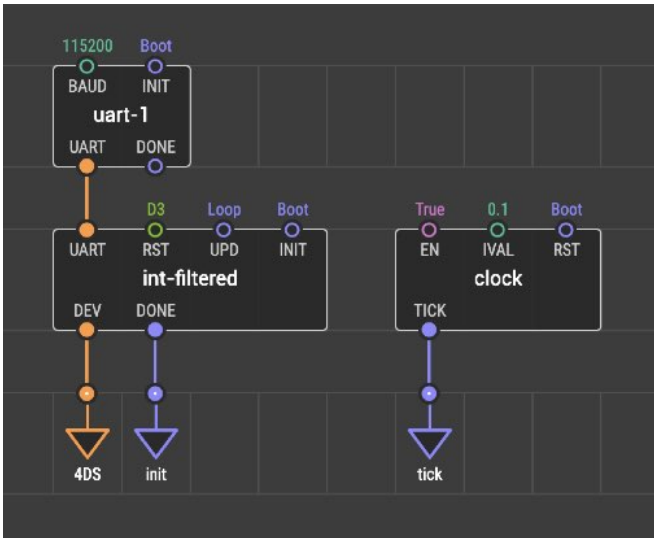
## Control system in XOD

The second major software component of the instrument design is the control of peripheral hardware from the Arduino-compatible Mega 2560 PRO Mini board. The control software was composed in XOD, and tested with open frame hardware chassis, before integration into prototype instruments. As described in this guide, XOD employs a visual dataflow model for composing Arduino-compatible code, with interactive debugging tools and ability to rapidly deploy control routines in simple and low-cost embedded systems...and accessible for users with no conventional programming skills. XOD programs are called patches, and they consist of a series of nodes that can represent physical devices or computational functions. Each node has pins that represent ports for input or output of data values or timing pulses. Input pins are located on the top of a node, and output ports are positioned at the bottom of the nodes. Programs are assembled by 'wiring' the node ports to create flows of data and timing through the program. The XOD program that is used to

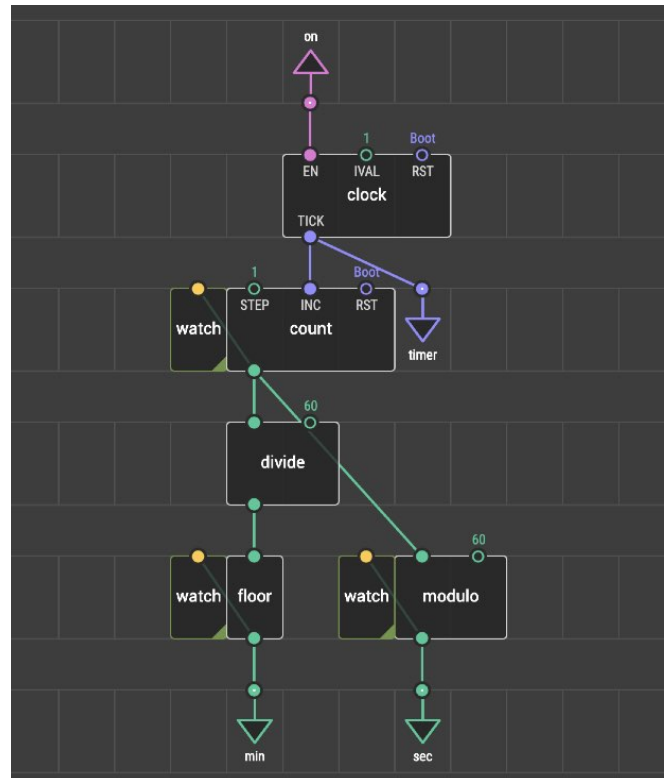




control the workings of the AirFlow device is shown above. At first glance this XOD patch looks complicated, but it can be broken down:



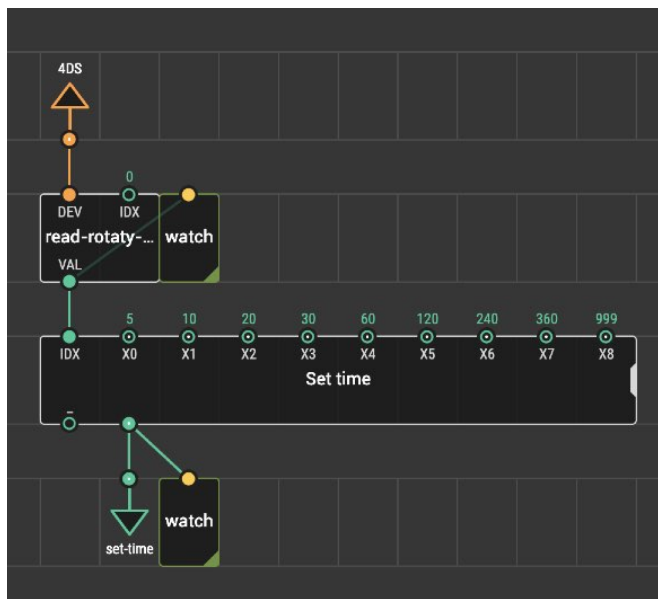
First, the connection to the 4D Systems touchscreen is defined - here connected to a hardware-based fast serial port (115200 baud **uart-1**). A fast clock for program operations is also defined (clock with 100 msec intervals). The arrows represent values that can be shared in a bus-like way with other nodes. For example, the fast clock node shares 100 msec pulses with other nodes through the tick bus.



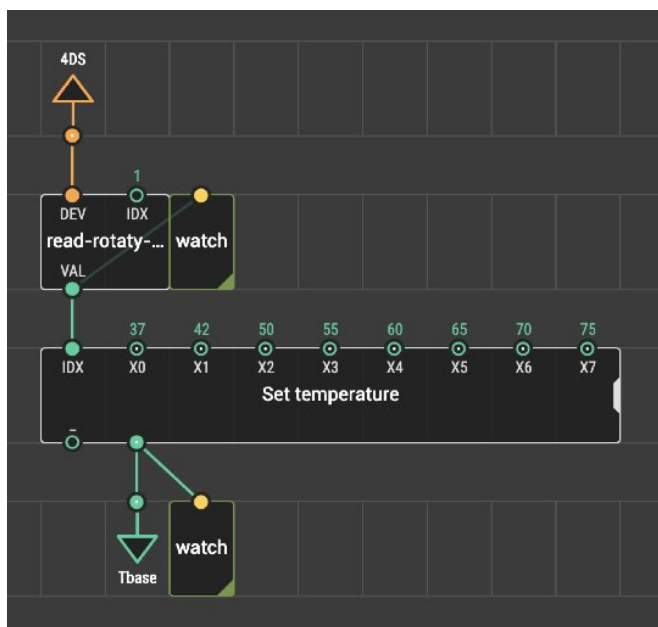
A second **clock** node is defined, which send pulses at 1 sec intervals. A count node is used to generate a running total of the

# AirFlow reactor prototype

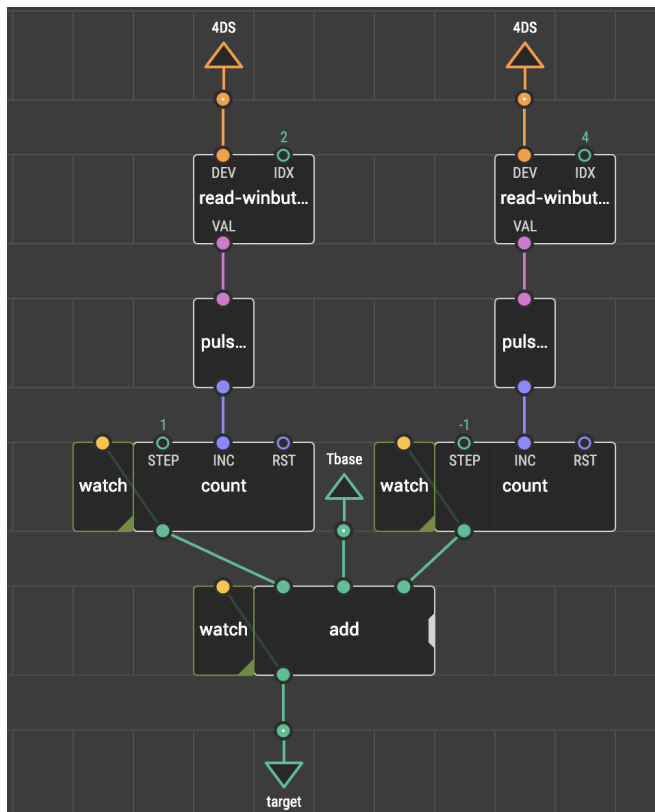
number of seconds elapsed. This value is sent to a **divide** node, divided by 60, and the **floor** node used to define the quotient (number of minutes elapsed), and the **modulo** node used to calculate the remainder (seconds between successive minutes elapsed). The min (minutes) and sec (seconds) values are shared with other nodes. (The **watch** nodes are used to show values in real-time during debugging with the hardware attached to the machine running the XOD development environment).



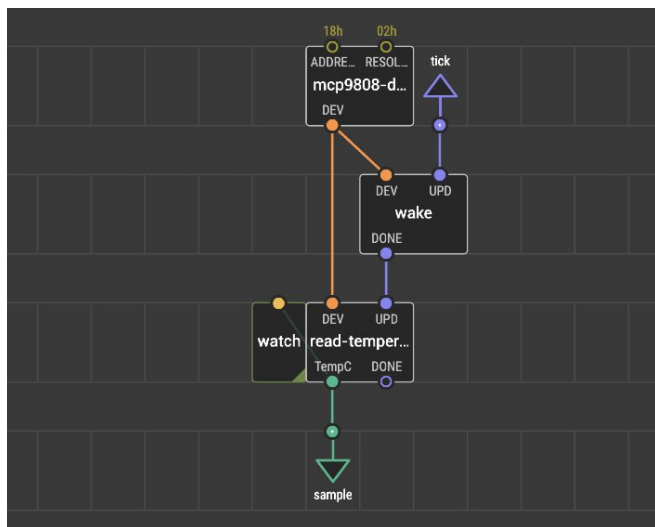
The touchscreen contains a widget that is used to set the time of a reaction. It is configured as a 9-position rotary switch which can be controlled by touch. The setting of this on-screen device is sent from the 4D Systems touchscreen to the microcontroller via the serial port that connects them. The **read-rotary-switch** node reads an index value from the switch (values from 0 to 8, depending on the chosen position). The **nth-input** node decodes this information and assigns a value corresponding to that displayed on the touchscreen controller, and shares this with other nodes as the time setting for the reaction time.



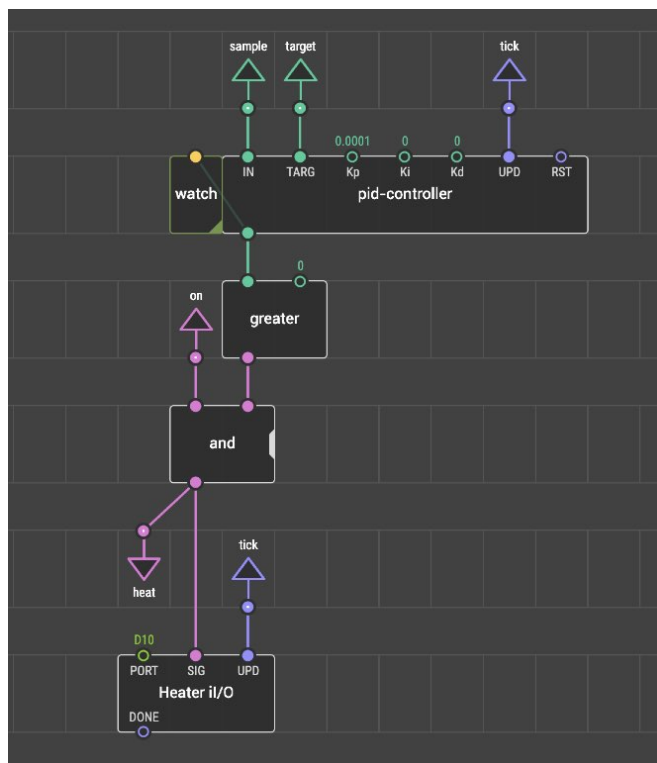
Similarly, a **read-rotary-switch** node is used to capture the index position for the rotary switch on the touchscreen. This is used to define the base setpoint for vessel heating, and is shared with other nodes as the value: Tbase.



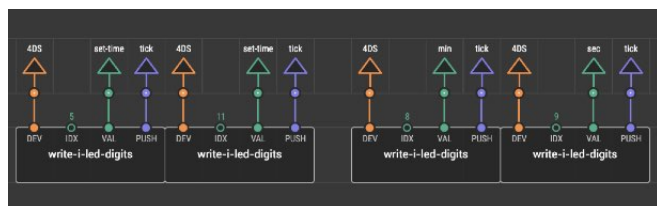
The base set temperature can be modified by two on-screen buttons that can be used to jog the set temperature up or down by 1°C. Two **read-winbutton** nodes monitor these momentary activated buttons. The boolean output of each button is fed to a **pulse-on-true** node which converts each button press to a pulse. Two separate count nodes are used keep a tally of the number of upward and downward button presses. These are then added to the base temperature setting, which is incremented or decremented accordingly - and the final temperature setpoint is shared as value: target.



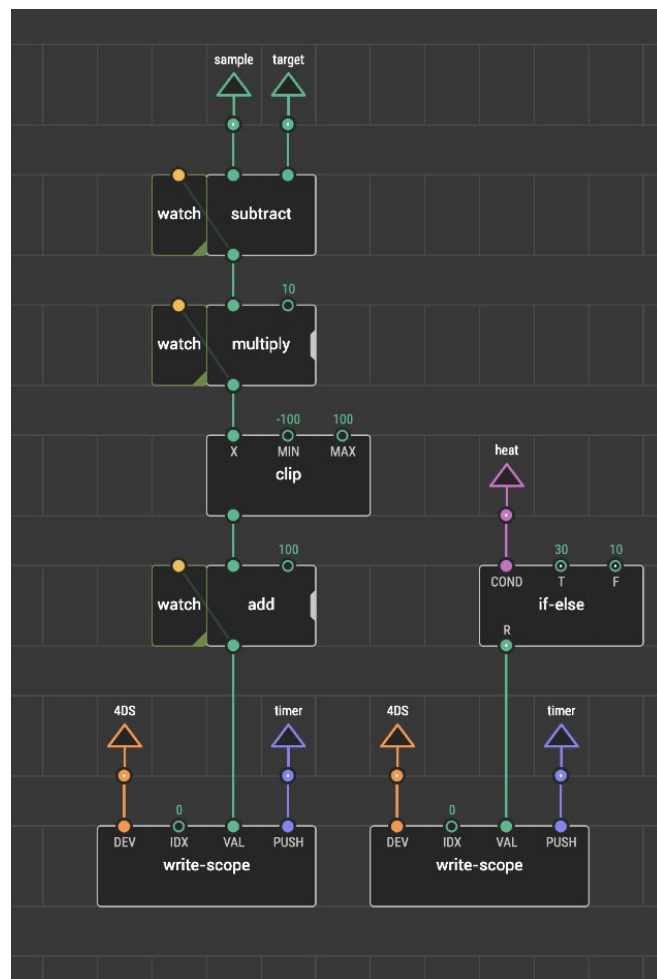
The current vessel temperature is read from the MCP9808 sensor. An external library (**wayland/mcp9808-thermometer**) provides special nodes for work with this hardware. The sensor is a sophisticated pre-calibrated device that is accurate to +/- 0.25°C between -40 and 125 °C. More information about the MCP9808 sensor and associated XOD nodes can be found in this book. The node **mcp9808-device** initialises the I2C device and **read-temperature** extracts the current temperature value, calibrated in °C. Reads are updated at 100msec intervals through use of a **wake** node, which is triggered by tick pulses.



XOD provides a node that incorporates code for a proportional-integral-derivative (PID) controller. The PID controller takes actual temperature and setpoint values, and calculates the best way to control an on-off heating circuit to rapidly reach and stabilise the temperature at the target temperature, based on three parameters, Kp, Ki and Kd. Here, a **pid** node is used to calculate whether to heat (+ve value) or not (-ve value). A **greater** node is used to check whether the output is above zero, and produce a boolean value. This is combined with the on/off switch signal through an **and** node - to activate the heater only when (i) a run has been activated, and (ii) called by the **pid** node. The heater is activated by switching a **digital-write** node (Heater I/O) connected to port (D10) that controls the MOSFET, controlling power to the heater element. In this device, control of the fan is simple, it is always working when the device is powered on. If the fan was to be controlled during the run, it would be advisable to program a cooling period after the end of a run to avoid any possibility of local overheating. One could also program additional visible or audible alerts at the end of reaction times, or for errors.



The device writes a number of values and parameters to different display widgets programmed into the touchscreen display. There are relatively large number of widgets on the different screens, and a corresponding large number of output nodes that send the relevant information to the appropriate screen element. A number of example are shown below, where values are sent to different, indexed digit displays via **write-i-led-digits** nodes. The values are updated every 100 msec due to triggering by tick pulses. Values are updated internal to the touchscreen controller, independent of whether they are currently displayed. A larger number of widgets and/or rate of updating can tax the rates of transmission through the serial port, causing errors or lockup, so fast serial hardware communication (or slower rate of updates) is needed.

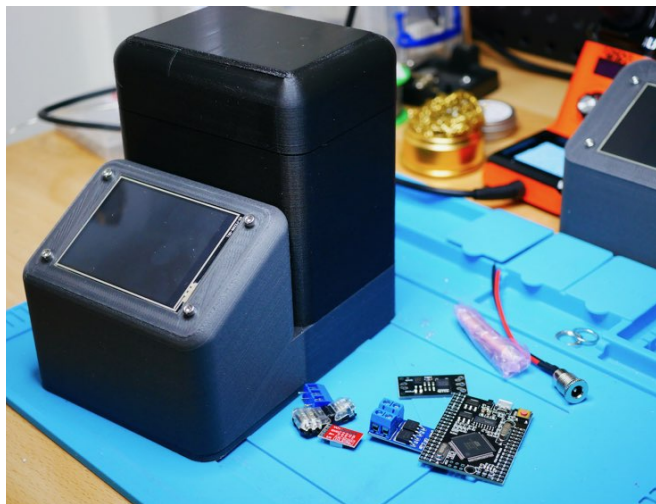


The "plot" screen on the touchscreen provides a historical view of vessel temperatures during a run. This is done by setting up a scope widget in Workshop4 when programming the touchscreen. The scope widget provides an oscilloscope-like display, where the size, X and Y scales and number/colour of traces can be preset in Workshop4. In XOD, instances of the **write-node** can be used to plot new data, and control the rate of updating. For example, in this case, the vessel temperature is plotted as a differential between the target and actual vessel temperatures. A **subtract** node is used to calculate the temperature difference. This value needs to be converted to a pixel value that is compatible with the on-screen scope widget (multiply by 10 and clip between -100 and 100, and add to offset by 100 pixels). At the same time, a signal corresponding to the state of heating is added to the plot. An **if-else** node is used to convert the boolean heater signal to numerical values (30 if true, 10 if false). The



# AirFlow reactor prototype

numerical values can be plotted as 2-state values on the scope widget (which can handle up to 4 traces). This allows the user to visualise the frequency and timing of heater activation, which can be useful for tuning the PID algorithm, and as a general indicator during normal use.

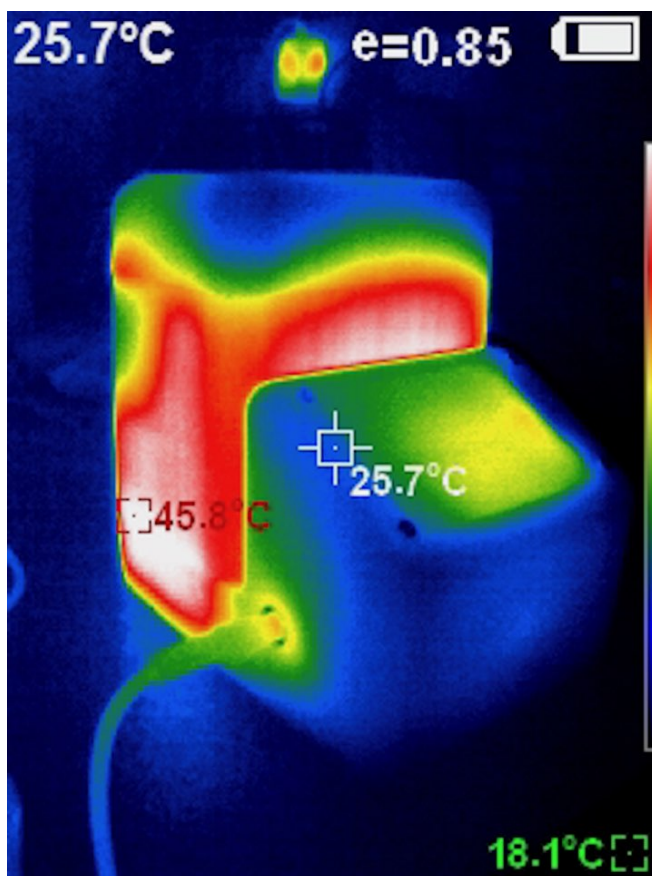
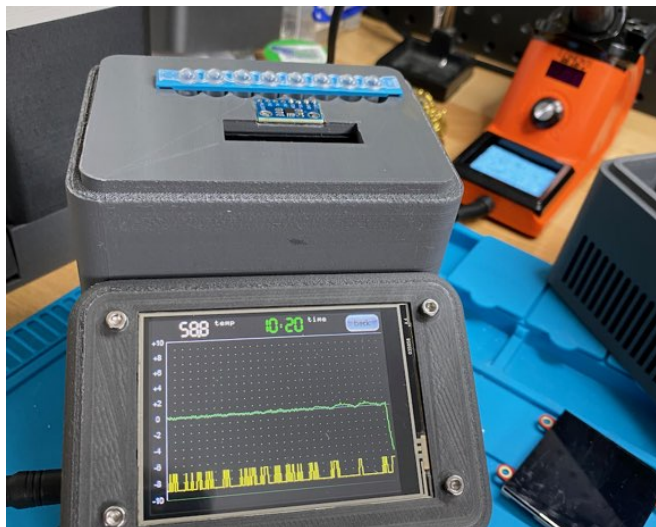


The 'code' for Workshop4 and XOD are both easy to modify and can be used to create customised interfaces.

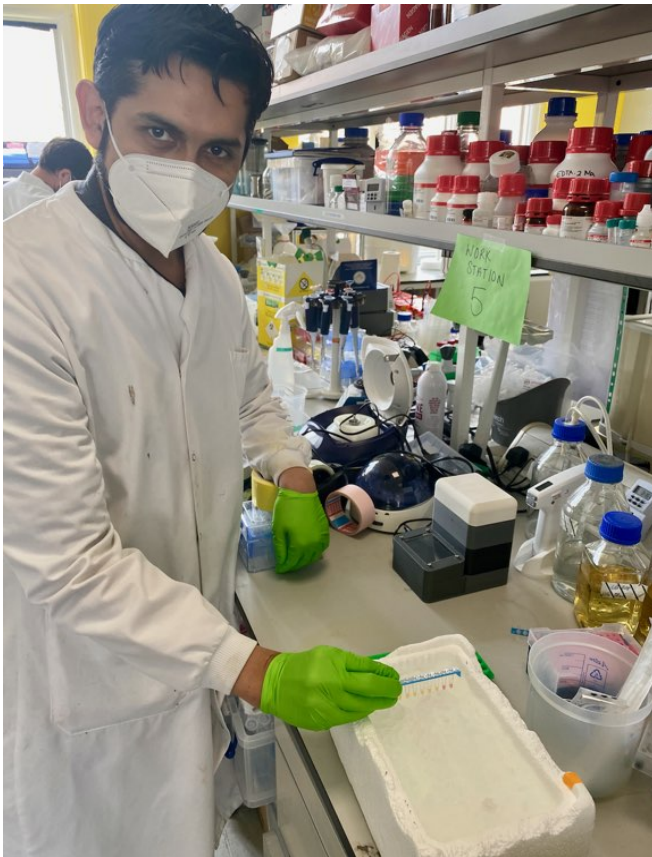
## Performance

The built microreactors have been through a number of design iterations ([hackster.io/jim-hasehoff](https://hackster.io/jim-hasehoff)), including early prototyping experiments detailed at the bottom of this page. The devices have proved relatively easy to construct, and due to the use of precalibrated sensors, simple to implement.

The vessels are generally run at around 65°C for LAMP reactions used for diagnostics. The reactors reach the working temperature within 2 minutes, and maintain the set temperature to within <1°C tolerance for the duration of the run. The hottest external surfaces of the AirFlow reactor (adjacent to the PTC heater) reach 45°C after prolonged running at 65°C, otherwise simply warm to the touch.



The AirFlow reactor has been tested successfully by running LAMP diagnostics experiments, conducted by Dr. Fernando Guzman Chavez in our lab in Cambridge. The reactions mixtures were assembled using New England BioLabs (NEB) WarmStart® Colorimetric LAMP 2X Master Mix, a kind gift from NEB.



### Costs

The current approximate retail cost of the components is around £100. The major costs are due to the touchscreen and 3D print materials. (COVID supply issues have elevated the cost of the TFT-LCD touchscreen). Cost reduction might centre on choice of cheaper display, and reducing the size and mass of the 3D printed vessel.

### Hardware components

- 50W, 12V PTC heater element
- ARX CeraDyna FW1275-A1041C centrifugal Fan 75 x 75 x 15mm, 16.58m<sup>3</sup>/h
- Adafruit MCP9808 I2C Temperature Sensor Breakout Board
- Mega 2560 PRO Mini Embedded MCU board
- 4D Systems gen4-uLCD-32DT
- D4183 MOSFET Trigger Switch Drive Module 400W
- LR7843 MOSFET switch module
- ElectroCookie Solderable Breadboard
- 5.5 x 2.1 MM 10A DC Power Jack Socket
- 4 Pin Cable with XH2.54mm pitch plug JST XH
- T-Tap 2-way quick splice wire connector
- Mini DC-DC 3A Buck Step-down Converter 12v to 5V
- Extruder GreenTEC Pro 3D printing filament
- Extruder GreenTEC Pro Carbon

### Software apps and online services

- Fusion 360
- Autodesk Fusion 360
- XOD
- Workshop4 IDE
- 4D Systems Workshop4 IDE

### Hand tools and fabrication machines

- Ultimaker S3 3D printer
- Dimafix pen, 100g
- UV curing liquid plastic glue
- Blu-tack

Jim Haseloff  
 University of Cambridge  
<https://www.hackster.io/jim-haseloff>

For more Biomaker projects, see:  
<https://www.hackster.io/biomaker>

### Future developments

- Continue refining the design of the vessel to reduce cost of materials and increase speed of printing. Investigate designs suitable for injection moulding.
- Explore different display options to reduce cost.
- Test different venting options to allow thermal cycling.
- Introduce optics in the manifold to allow integration of quantitative imaging and analysis.
- Add wireless communications and web-based data collection.



# Additional Information

## Alternative Development Boards

The Grove board is a great place to start with building your own devices, as it is simple to use, low-cost, easily accessible, and comes with a range of useful inbuilt components. However, there are a wide variety of other boards available for getting started with projects like this.

The two most commonly used types of development boards are Arduino and Raspberry Pi, and each of these companies provide a range of boards for different uses. Whilst Arduino boards are microcontrollers that can perform one programme at a time, Raspberry Pi boards are fully-operational computers that can perform multiple tasks at once. A Raspberry Pi may be better for more complex projects, but Arduino boards are easier to use and suited for most simple projects. Note that XOD does not yet support programming of Raspberry Pi boards.

You can find a comparison of Arduino models on the SparkFun website ([www.learn.sparkfun.com/tutorials/arduino-comparison-guide](http://www.learn.sparkfun.com/tutorials/arduino-comparison-guide)) and a comparison of Raspberry Pi models on the PiHut website ([www.thepihut.com/blogs/raspberry-pi-roundup/raspberry-pi-comparison-table](http://www.thepihut.com/blogs/raspberry-pi-roundup/raspberry-pi-comparison-table)).

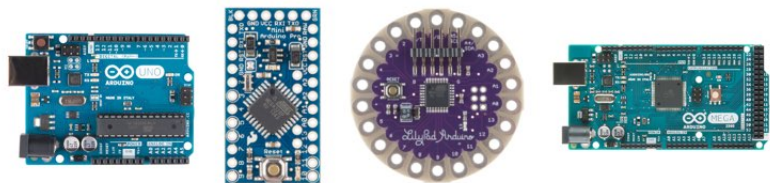


Image Credits: SparkFun Electronics CC

Arduino boards, left to right: Arduino Uno, Arduino Pro Mini, Lilypad Arduino, Arduino Mega 2560

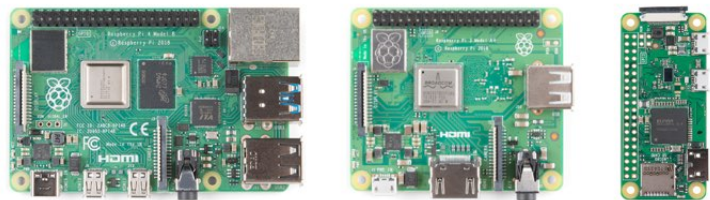


Image Credits: SparkFun Electronics CC

Raspberry Pi boards, left to right: Raspberry Pi 4 Model B, Raspberry Pi 3 A+, Raspberry Pi Zero W

## Useful Links

<b>BIOMAKER</b>	<a href="http://www.biomaker.org">www.biomaker.org</a> Collection of technical information, pointers to tutorials and software resources, information about the Biomaker Challenge
<b>NO-CODE PROGRAMMING</b>	<a href="http://www.biomaker.org/nocode-programming-for-biology-handbook">www.biomaker.org/nocode-programming-for-biology-handbook</a> Information on the No-Code Programming for Biology programme, handbook downloads, tutorials and videos.
<b>HACKSTER</b>	<a href="http://www.hackster.io/biomaker">www.hackster.io/biomaker</a> Biomaker community hub used for open documentation of Biomaker projects and tutorials.
<b>XOD</b>	<a href="http://www.xod.io">www.xod.io</a> Download XOD software, libraries, documentation and forum advice.
<b>ARDUINO</b>	<a href="http://www.arduino.cc">www.arduino.cc</a> Official repository of Arduino information.
<b>ARDUINO CREATE</b>	<a href="http://www.create.arduino.cc">www.create.arduino.cc</a> Integrated resource for code and project-sharing.
<b>SEEED STUDIO</b>	<a href="http://www.seeedstudio.com">www.seeedstudio.com</a> Hardware supplier for the Biomaker Starter Kit and Grove components.
<b>OPEN SMART</b>	<a href="http://www.open-smart.aliexpress.com/">www.open-smart.aliexpress.com/</a> Source of hardware for Biomaker expansion kit.
<b>SPARKFUN</b>	<a href="http://www.sparkfun.com">www.sparkfun.com</a> Good source of practical information about microcontrollers and devices.
<b>ADAFRUIT</b>	<a href="http://www.adafruit.com">www.adafruit.com</a> Good source of practical information about microcontrollers and devices.
<b>INSTRUCTABLES</b>	<a href="http://www.instructables.com/classes/">www.instructables.com/classes/</a> Classes in many maker skills, including electronics and 3D printing.
<b>FRITZING</b>	<a href="http://www.fritzing.org">www.fritzing.org</a> Open source circuit layout and illustration.
<b>PROCESSING</b>	<a href="http://www.processing.org">www.processing.org</a> Software sketchbook for dynamic graphics and visual arts.
<b>ENGINEERING BIOLOGY IRC</b>	<a href="http://www.engbio.cam.ac.uk">www.engbio.cam.ac.uk</a> Information, news and events from the Engineering Biology Interdisciplinary Research Centre at the University of Cambridge
<b>OPENPLANT</b>	<a href="http://www.openplant.org">www.openplant.org</a> Information, news and events from the BBSRC-EPSRC Synthetic Biology Research Centre

# Glossary

<b>ACCELEROMETER</b>	Accelerometers measure the acceleration of an object, i.e. any change in velocity (speed and direction). A 3-axis accelerometer, like the one included in the Grove board, can sense when the board is moved or tilted in any direction (X, Y and Z axes).
<b>ACTUATOR</b>	Actuators are output devices that convert electronic signals into mechanical movement. For example motors, belts or pumps.
<b>ADAFRUIT</b>	Adafruit Industries is an open-source hardware company that provides electronic components, tools, accessories and learning resources. Their components are compatible with Arduino and Raspberry Pi hardware.
<b>ANALOG</b>	Analog signals, unlike digital signals, are continuous and can take an infinite number of values. Analog devices measure continuous variables, such as sound or light intensity. Many environmental sensors are analog devices. Computers use digital, rather than analog signals, so analog signals must first be converted to digital signals by the microcontroller.
<b>ARDUINO</b>	Arduino is an open-source electronics company. They make openly available programming software and low-cost hardware to allow anyone to get started making their own interactive electronics projects.
<b>ARDUINO IDE</b>	The Arduino Integrated Development Environment (IDE) is Arduino's free software for programming Arduino boards using the C++ programming language. It is an alternative to the XOD IDE, and can be used alongside XOD (see p79).
<b>ARDUINO UNO</b>	The Arduino UNO was the first USB-based Arduino board, consisting of a microcontroller chip, printed circuit board (PCB) and a series of digital and analog input-output pins to connect shields and external hardware. The Arduino UNO R3 is the third revision of this board, and is what the Seeduino and Grove board are based upon.
<b>ATMEGA328P</b>	The Atmega328P is the microcontroller chip used in the latest versions of the Arduino board, including the Grove board.
<b>BAROMETER</b>	A barometer device measures air pressure, and can therefore be used to monitor or forecast weather, or to measure altitude.
<b>BIOMAKER</b>	Biomaker is an initiative founded at the University of Cambridge that focuses on training and providing funding and resources for researchers interested in the intersection of biology, engineering and computing. Biomaker activities include the annual Biomaker Challenges and No-Code Programming for Biology training.
<b>BREADBOARD</b>	Breadboards are simple devices for developing and prototyping electrical circuits, without the need for soldering. They consist of rows of sockets that are connected via electrical wiring. Components can be plugged directly into these sockets using wires or metal pins. Any devices plugged into the same row of sockets will be connected together.
<b>BREAKOUT BOARDS</b>	Breakout boards are used to make wiring of electronic components easier. They usually consist of a small PCB board with a single, or small number of electronic components attached. For example, an LED or OLED Screen breakout board. They can be easily attached to a development board via wires, pins and sockets, or plugs.

<b>BUS (XOD) parts</b>	In computing a bus is a communication system that transfers information between computers, or between different parts of the same computer. In XOD, we can use buses to transfer information between one part of our patch and another, without having to connect them via links. This is done using the <i>to-bus</i> and <i>from-bus</i> nodes to send information to, and receive information from a particular bus.
<b>COMMUNICATION and PROTOCOL</b>	A communication protocol is the method by which two or more electronic components exchange data. Ethernet, wi-fi bluetooth are all examples of communication protocols. Different components use different communication protocols (e.g. analog, digital, I2C) and so will need to be connected to the Arduino board in different ways.
<b>DEBUGGER (XOD)</b>	The Debugger is XOD's simulator function. It can be used to simulate your patch, or to edit and 'debug' your patch after upload to the board. You can start the debugger using the 'Simulate' button, or using the 'Upload and Debug' button to upload the patch at the same time. In debugger mode, you can use tweak and watch nodes to edit and monitor your patch in real-time.
<b>DEVELOPMENT BOARD</b>	A microcontroller development board, like the Grove Arduino board, houses a microcontroller chip on a small PCB board along side some additional parts and connections making it easy for anyone to programme and connect components to a microcontroller at home. Development boards are intended to be cheap and easily accessible, and are often used for developing prototypes and custom instruments.
<b>DEVELOPMENT HOST development</b>	A development host is the device you will use to write and develop the programme you want to install on the board. To programme the Grove board you will use a laptop or PC as the development host.
<b>DIGITAL</b>	Digital signals, unlike analog signals, can only take finite and discrete values. For example, an LED can be 'on' or 'off'. Digital signals can be made to behave in a similar way to analog signals, for example, you can change the brightness of an LED, but ultimately there are a finite and discrete number of values that the brightness of an LED can take. Computers use digital signals, and most electronic components are digital. For example, screens, buttons, and some types of sensor.
<b>GROVE</b>	Grove is toolkit of easy-to-use Arduino-compatible electronics. It uses a 'plug-and-play' system of modules that can be easily fitted together to build custom devices. It is developed by the company Seeed Studio.
<b>HACKSTER</b>	Hackster is an online platform for recording and sharing electronics projects. It provides a simple way to document and browse projects, and has a large community of contributors, including companies such as Arduino and Seeed Studio.
<b>HEADER SOCKETS</b>	The header sockets (also known as female headers) are connectors that are wired to the PCB board and provide "female" sockets. They give us a way to easily connect external components to the board, either via male-to-male hook-up wires, or via an expansion shield.
<b>HOOK-UP WIRES</b>	Hook-up wires (also known as jumper wires or jumper cables) are used to connect components to the Arduino board and come in several different types. Female-to-female hook-up wires have connector sockets at each end that plug into metal pins on components, on the Arduino board, and on shields. Male-to-male wires, which have metal pins on each end that fit into female sockets, header sockets or breadboards. Male-to-female wires that have a female socket at one end and a male pin at the other end. Hook-up wires can also come pre-fitted with plugs to fit into compatible sockets. For example Grove plugs or Open Smart (JST-XH) plugs.
<b>HYGROMETER</b>	Hygrometer devices are used to measure humidity, i.e. the amount of water vapour present in the air or in soil.
<b>I2C</b>	Inter-integrated circuits (I2C) are a digital communication protocol used to communicate with multiple devices at once. With I2C communication several devices can be connected to the same pin of the microcontroller, and each device is given a "name" digitally (known as an address). Addresses are written as XXh, with XX being a two digit code of numbers and letters. For example 19h or 3Ch.
<b>INSPECTOR (XOD)</b>	In XOD, the Inspector pane is the place where a nodes can be edited. For example, you can change the parameters of a node's pins, change the name of a node, or add a description. You must click on a node in the patch for these options to appear. The Inspector pane appears on the left hand side of the screen below the Project Browser pane, and can be toggled on and off using the slider bar button in the top left, or by navigating to 'View > Toggle Inspector' in the menu bar.
<b>LED</b>	A light-emitting diode (LED) is a bright, low-power light source that generates light by passing a current through a semiconductor diode.

# Glossary

<b>LIBRARY (XOD)</b>	In XOD (and in other coding software such as Arduino), libraries are collections of ready-to-use nodes (or code). They are often designed to help you use a specific piece of hardware (e.g. wayland/bmp280-barometer) or as a collection of nodes with similar functions (e.g. xod/math). The XOD IDE has several libraries pre-installed, but you can add more libraries using the 'Add Library' button (books with a + symbol, in the Project Browser) or by navigating to 'File > Add Library...' in the menu bar.
<b>M5STACK</b>	M5Stack is a hardware company which provides its own wi-fi and bluetooth enabled development system, as well as a series of Grove-compatible components called 'units'.
<b>MICROCONTROLLER</b>	A microcontroller is a small low-power computer embedded into a device. In contrast to a general purpose computer like a laptop or PC, microcontrollers are often designed to complete one task and run one specific programme. The Grove Arduino board contains a reprogrammable microcontroller so you can upload your own programme on to the board and create your own devices.
<b>NO-CODE PROGRAMMING</b>	Node-code programming is an increasingly popular mechanism allowing people to programme hardware and software using a graphical user interface, rather than requiring them to learn and write text-based code. The Biomaker No-Code Programming for Biology initiative has adopted no-code and low-code programming as a way to train biologists, and others with little or no coding experience, to build their own custom devices.
<b>NODE (XOD)</b>	In XOD nodes are used as "a visual representation of a physical device or function". They can represent an electronic component, a mathematical function or any number of other functions that a computer can perform. They appear on a patch as a dark grey box outlined in white, with the name of the node printed in the middle. They may have small coloured circles (pins) on the top and bottom which represent inputs and outputs.
<b>OLED SCREEN</b>	Organic light emitting diode (OLED) screens are an alternative to LCD screens used mainly for TVs. Instead of having a backlight to illuminate pixels, each pixel can produce its own light. This can improve contrast.
<b>OPEN SMART</b>	OpenSmart is a group of technology companies interested in the production and development of open-source hardware. They are based in Shenzhen, China. Open Smart components are used in the Biomaker Expansion Kit.
<b>PATCH (XOD)</b>	In XOD a patch is the working area in which a programme is built. It is similar to a document or source file in other systems, but instead of text code the patch is built with nodes.
<b>PCB</b>	A printed circuit board (PCB) is composed of a thin fibreglass board with conductive tracks of copper etched on the surface or between the layers. They are used to connect electrical components, which are usually soldered onto the board.
<b>PHOTORESISTOR</b>	A photoresistor or light dependent resistor (LDR) is a component that is sensitive to light. When light falls upon it then the resistance changes, and this change is used as an electronic signal.
<b>PIEZOELECTRIC BUZZER</b>	Piezo buzzers are simple devices that can generate basic beeps and tones. They work by using a piezo crystal, a special material that changes shape when voltage is applied to it. If the crystal pushes against a diaphragm, it can generate a pressure wave which the human ear picks up as sound.



<b>PIN (ARDUINO)</b>	In electronics, "pin" is used to refer to the electrical contacts on a component, i.e. the parts of a component that are used to connect to other components. On the Grove board, the microcontroller chip has a number of pins that are connected to both the components on the board, and to the Grove sockets and header sockets of the central module, which allows them to communicate with additional components. In XOD the Arduino pins are referred to as "Ports" to avoid confusion with XOD pins.
<b>PIN (XOD)</b>	In XOD "pin" is used to refer to the inputs and outputs associated with a specific node. They appear as small round circles on the top (input pins) and bottom (output pins) of a node. Pins are coloured according to their data type.
<b>POTENTIOMETER</b>	Potentiometers (often shortened to "pot") are variable resistors that allow you to alter the resistance, and therefore the current flowing through a circuit, without the need to reprogram the device. They are often found in the form of a knob, slider or screw.
<b>PROJECT BROWSER</b> Under (XOD)	The XOD Project Browser is where you will find the current project you are working and libraries you have installed. The 'My Project/[name of your project]' dropdown you will find all of the patches (or files) in your project. Below that is a list of libraries. Clicking on the dropdown button of a library will allow you to browse the nodes in that library. Dragging a node from the Project Browser into the patch will add that node to your patch. At the top of the Project Browser are the 'New Patch' and 'Add Library' buttons. The Project Browser pane appears on the top left of the screen, and can be toggled on and off using the hub button in the top left, or by navigating to 'View > Toggle Project Browser' in the menu bar.
<b>QUICK HELP (XOD)</b>	The Quick Help pane in XOD is where you can find information about a node and it's pins. When you click on a node information about that node and it's pins will appear in the Quick Help pane. The Quick Help pane appears on the top right of the screen, and can be toggled on and off using the question mark button in the top right, or by navigating to 'View > Toggle Quick Help' in the menu bar.
<b>SEED STUDIO</b>	Seed Studio is an open-source hardware company. They developed the Seeduino Lotus development board (based on the Arduino Uno R3 development board) and the Grove system of components which use plugs to easily connect modules.
<b>SHIELD</b>	Shields are modular circuit boards that piggyback onto your Arduino to instil it with extra functionality. Shields can have specific functions, such as a wifi shield that will allow your board to transfer information via wifi, or can have more general functions, like an expansion or prototyping shield. These allow you to easily connect any number of custom components. Some shields can be stacked on top of one another to create combinations of modules and functions.
<b>SPARKFUN</b>	SparkFun is an open-source hardware company that provide a range of development boards and components, as well as tutorials and learning resources on programming, electronics and working with hardware.
<b>TERMINALS (XOD)</b>	XOD terminal nodes (input and output nodes) are used to allow a patch to communicate with 'the-outside-world'. Adding terminal nodes to a patch will allow that patch to be used as a node in other patches, with the names and types of the terminals corresponding to the names and types of the pins on your new node.
<b>USB DRIVER</b>	A USB driver is a piece of software that allows your computer's operating system to communicate with external hardware, such as a hard drive or a microcontroller development board like the Grove board. The Grove board uses the CP210 driver from Silicon Labs, and you may need to download this driver in order to use your board.
<b>XOD</b>	XOD is an open-source software company that provides the the XOD Integrated Development Environment (IDE). The XOD IDE is free software that allows you to programme Arduino-based development boards using visual programming rather than text-based coding. XOD software uses graphical nodes to represent functions, and nodes are connected together to visualise data flow and programme hardware.

# Index

## A

accelerometer 7, 56, 58–59, 80  
 AirFlow 176, 179, 181, 185, 188  
 Arduino 3, 6, 8, 20, 72, 94, 155–156, 171, 190  
 ATmega328P 8  
 Arduino IDE 155–156, 158  
 Arduino libraries 41, 156

## B

Biomaker iii, 4, 90, 191, 198  
   initiative iii, 192  
   www.biomaker.org iii, 4, 19, 87–88, 90,  
     155–156, 191  
   www.hackster.io/biomaker 5, 91, 191  
 biophotovoltaics 172  
 breadboard 86, 90, 95, 99, 111, 119, 123  
 buzzer 6, 26, 29, 64, 72

## C

C++ 155–156, 160, 166, 171  
 capacitive soil moisture sensor 150, 152  
 colour sensor 118, 120  
 communication 10, 94, 96, 180, 187  
   analog 10, 94, 97  
   digital 10, 97  
   I2C 10, 87, 97, 187  
 components  
   1602 liquid crystal display 98  
   BH1750 114  
   BMP280 43, 73, 80  
   DHT11 78  
   DHT20 79  
   DS1302 106  
   HC-SR501 130  
   HX711 138  
   LIS3DHTR 80  
   MCP9808 126, 129, 180  
   nRF8001 bluetooth 172  
   potentiometer 26  
   RCWL-0516 134  
   SGP30 172  
   SHT20 110  
   TCS3472 118  
   TDS Meter V1.0 142  
   TSL2591 157  
   VL6180X 122  
   WS2812 RGB LED 102

## E

encapsulation 153

## F

Fritzing 90, 191  
   https://fritzing.org 90

## G

Grove 3, 7, 10, 14, 72, 85–87, 90, 96  
 connectors 96

## H

Hackster 5, 91, 173–174, 189, 191  
 hygrometer 7, 34, 73, 78

## L

laser Range Finder 122  
 LED 6, 20, 22, 38, 102, 119  
 light sensor 73, 77, 114, 116, 156

## M

microcontroller 2, 6, 8, 10, 180, 182  
 motion sensor 130, 132, 135

## N

No-Code programming iii, 4, 84, 191

## O

OLED 6, 50, 52, 54–55, 58, 72, 75  
 Open Smart 84, 86, 88, 191  
 OpenPlant iii, 191, 198

## P

pin 11, 14  
 port 11, 15  
   conflicts 88  
 publishing libraries 55  
 PWM 15, 74, 94

## Q

Qwiic 156–157

## R

radar proximity sensor 134  
 real time clock (RTC) 106

## S

Seed Studio iii–iv, 6, 85, 96, 145, 191, 193  
   Grove Beginner Kit 6, 20, 74, 87, 92  
   software expansion 155  
 SPI 94

## **T**

temperature sensor           **79–80, 110, 126, 128, 180**  
Totem                               **92, 99**

## **U**

UKRI   **iii**  
University of Cambridge           **iii, 191–192, 198**  
USB driver                                       **18**

## **W**

water level sensor                       **146**  
water quality sensor                   **142**  
waterproof                                   **153**  
weight sensor                               **138**

## **X**

XOD           **iv–v, 2–5, 12, 15, 18, 20, 32, 37, 43, 45, 48, 72, 74, 155–156, 171, 184**  
XOD forum                                   **41**

# Acknowledgements

## AUTHORS

Jim Haseloff  
Stephanie Norwood  
Matt Wayland

## IMAGES

Jim Haseloff  
Stephanie Norwood  
Matt Wayland  
Seed Studio  
SparkFun Electronics Inc.  
Adafruit Industries LLC  
Biomaker Challenge Participants

## DESIGN & LAYOUT

Stephanie Norwood  
Jim Haseloff

## HARDWARE

Seed Technology Co. Ltd.  
Arduino S.r.l.  
Adafruit Industries, LLC  
M5stack-store  
OpenSmart Tech.  
Spark Fun Electronics Inc.

## SOFTWARE

XOD  
Arduino  
Hackster,

## XOD LIBRARY CREATORS

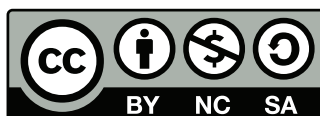
Matt Wayland  
Marco Aita  
Cesar Sosa  
XOD user: gst  
XOD user: e  
XOD user: gweimer

## FUNDING AND SUPPORT

Biotechnology and Biological Sciences Research Council (BBSRC)  
Engineering and Physical Sciences Research Council (EPSRC)  
Natural Environment Research Council (NERC)  
Global Challenges Research Fund UK  
OpenPlant Synthetic Biology Research Centre  
Department of Plant Sciences, University of Cambridge  
Engineering Biology Interdisciplinary Research Centre (EngBio IRC) University of Cambridge  
Seed Studio

## CONTACT

Jim Haseloff  
jh295@cam.ac.uk



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.







Designed for those with little to no experience working with coding or hardware, this guide makes use of free open-source software and low-cost hardware to introduce you skills required for rapid prototyping of custom scientific instruments.

Learn how to:

- Understand and control an Arduino-based microcontroller
- Programme without using text-based code
- Use simple electronic devices such as screens and sensors
- Build your own devices for use in biological research

[www.biomaker.org](http://www.biomaker.org)



ISBN 978-1-7394029-0-7

